TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANTON MURAVEV

EFFICIENT VECTOR QUANTIZATION FOR FAST APPROXIMATE NEAREST NEIGHBOR SEARCH

Master of Science thesis

Examiners:
Prof. Moncef Gabbouj
Dr. Alexandros Iosifidis

Examiner and topic approved in the Faculty Council meeting of Computing and Electrical Engineering Faculty
on 4th May 2016

ABSTRACT

ANTON MURAVEV: Efficient Vector Quantization for Fast Approximate Nearest Neighbor Search
Tampere University of Technology
Master of Science Thesis, 49 pages, 0 Appendix pages
June 2016
Master's Degree Programme in Information Technology
Major: Data Engineering
Examiners: Prof. Moncef Gabbouj, Dr. Alexandros Iosifidis

Keywords: approximate nearest neighbor search, product quantization, additive quantization

Increasing sizes of databases and data stores mean that the traditional tasks, such as locating a nearest neighbor for a given data point, become too complex for classical solutions to handle. Exact solutions have been shown to scale poorly with dimensionality of the data. Approximate nearest neighbor search (ANN) is a practical compromise between accuracy and performance; it is widely applicable and is a subject of much research.

Amongst a number of ANN approaches suggested in the recent years, the ones based on vector quantization stand out, achieving state-of-the-art results. Product quantization (PQ) decomposes vectors into subspaces for separate processing, allowing for fast lookup-based distance calculations. Additive quantization (AQ) drops most of PQ constraints, currently providing the best search accuracy on image descriptor datasets, but at a higher computational cost. This thesis work aims to reduce the complexity of AQ by changing a single most expensive step in the process – that of vector encoding. Both the outstanding search performance and high costs of AQ come from its generality, therefore by imposing some novel external constraints it is possible to achieve a better compromise: reduce complexity while retaining the accuracy advantage over other ANN methods.

We propose a new encoding method for AQ – pyramid encoding. It requires significantly less calculations compared to the original "beam search" encoding, at the cost of an increased greediness of the optimization procedure. As its performance depends heavily on the initialization, the problem of choosing a starting point is also discussed. The results achieved by applying the proposed method are compared with the current state-of-the-art on two widely used benchmark datasets – GIST1M and SIFT1M, both generated from a real-world image data and therefore closely modeling practical applications. AQ with pyramid encoding, in addition to its computational benefits, is shown to achieve similar or better search performance than competing methods. However, its current advantages seem to be limited to data of a certain internal structure. Further analysis of this drawback provides us with the directions of possible future work.

PREFACE

The work presented in this thesis was carried out at the Department of Signal Processing, Tampere University of Technology (TUT) over a period of 7 months – from October 2015 till May 2016. Throughout this period the author was employed as a research assistant in the MUVIS group.

I would like to express my utmost gratitude to professor Moncef Gabbouj for an opportunity to work as a part of the group and for his invaluable advice, which will undeniably prove useful in my future scientific career. I am very grateful to my supervisor Mr. Ezgi Can Ozan for providing such an interesting topic to work with and for fruitful discussions, which motivated many positive changes in the proposed methods. I would also like to thank Dr. Alexandros Iosifidis for his guidance and important feedback throughout the work.

Finally, I offer my sincere gratitude to my parents. Despite the great distance between us, their support and encouragement were immense throughout the whole period of my studies, including the thesis preparation.

Tampere, June 2016

Anton Muravev

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ADC	Asymmetric Distance Computation
ANN	Approximate Nearest Neighbor
APQ	Additive Product Quantization
AQ	Additive Quantization
CQ	Composite Quantization
ICM	Iterated Conditional Modes
ITQ	Iterative Quantization
L-BFGS	Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm
LOPQ	Locally Optimized Product Quantization
LSH	Locality Sensitive Hashing
MRF	Markov Random Field
MSE	Mean Squared Error
NN	Nearest Neighbor
OPQ	Optimized Product Quantization
OTQ	Optimized Tree Quantization
PCA	Principal Component Analysis
PQ	Product Quantization
RPE	Residual Pyramid Encoding
SDC	Symmetric Distance Computation
SIFT	Scale Invariant Feature Transform
SQ	Stacked Quantization
TQ	Tree Quantization
t-SNE	t-distributed Stochastic Neighbor Embedding
VQ	Vector Quantization

c	codevector
C	codebook matrix
D	dimensionality of the data
H	search depth parameter for encoding
k	number of nearest neighbors to search for
K	number of codevectors per codebook
L	number of training iterations
M	number of codebooks
N	number of database (or training set) vectors
q	query vector
R	rotation matrix
x	database vector
$\tilde{x}$	reconstruction of database vector x
X	data matrix

1. INTRODUCTION

The nearest neighbor (NN) search, or the problem of matching the given object to the most similar one from the given group, is extremely common; it is not only specific to science or engineering, but permeates the everyday life in many forms, such as recognition. Its technical definition can vary based on the nature of the objects and how their similarity is established. The latter is usually defined as a function, providing a numeric output value. The objects of search are commonly vectors, allowing for a variety of similarity measures to be used, both metric and nonmetric.

While the search problem is trivial when the number of objects to consider is small, the advances in computer science and technology lead to a consistent growth of data sizes. For this reason the data structures allowing efficient search were extensively studied since 1970s [1]. Branch-and-bound approach in particular resulted in numerous types of search trees, allowing for queries to be of logarithmic complexity with respect to data size [2]. Space partitioning would remain dominant for decades thereafter.

The emergence of Big Data has led to reconsideration of many previously established solutions [3][4]. This new environment presents a number of challenges, one of which is the sheer volume of the databases. Traditional algorithms and methods commonly become computationally infeasible in such scenarios, sometimes to the point of complete inapplicability. The nearest neighbor search techniques were no exception from this; many well-established data structures were found to lose their advantages completely in higher-dimensional spaces, becoming inferior to exhaustive calculations [5].

Many current practical applications involving the search do not require perfectly accurate results. For instance, when information retrieval is performed, the returned documents or images are often deemed acceptable if they are relevant; the exact definition of relevance varies on case by case basis, but can generally be taken to mean “similar enough to the perfect outcome”. This factor, combined with previously mentioned computational costs issue, leads to the notion of the approximate nearest neighbor (ANN) search, which has been the focus of much recent research.

The focus of approximate search techniques is as much on accuracy as it is on scalability and low costs, both in computation and in memory use. Removing the requirement for an exact solution allows for a considerable complexity reduction, making the approximate search preferable in many practical scenarios. Some of the current large-scale applications include:

- information and content-based retrieval (e.g. VisualRank, as used in Google Image Search) [6][7];
- large-scale cluster analysis, as many common algorithms (e.g. k-means, hierarchical clustering) require NN search as an intermediate step [8];
- computer vision [9];
- recommendation systems [10];
- pattern recognition and classification [11][12][13].

Locality sensitive hashing (LSH) introduced a breakthrough in ANN search algorithms, leading to an emergence of the large family of hashing-based techniques [14]. All of these produce binary codes from the data, and Hamming distance between the codes is used as a substitute for the original similarity measure. The hash function is required to produce similar codes for similar data inputs, and vice versa. The exact choice of the hashing function naturally depends on the similarity measure used. Later algorithms attempted to learn suitable hash functions from the data itself, instead of just using the static formulations. Due to extensive research the properties of LSH and its derivatives are relatively well-understood. There is a wide variety of hash functions for many similarity measures and use cases. Due to these factors and typically modest computational costs, hashing-based techniques have been widely applied in practice and still draw the academic attention [7][14][15].

The drawback of this family of methods is the relatively poor search accuracy. In addition to the imperfection of the hash functions and the locality criterion, the binary codes simply lack the representation power to preserve the original pairwise distances. While reducing (compressing) the data does provide the approximate search benefits, it could be more efficient, if the same similarity measure could be retained after the transformation. Under the assumption of Euclidean distance (which often holds in practice) the data can be compressed (quantized) with an algorithm known as vector quantization (VQ), which operates completely in the original space [16].

While ANN search can be performed with VQ, it is not a practical solution. VQ is susceptible to a "curse of dimensionality", leading to exponential growth in calculations and memory requirements as the data grows in dimensions [17]. However, other techniques that avoid the issue have been derived in recent years, resulting in a quantization-based methods forming a family of their own [18]. These methods often enjoy a strong advantage over state-of-the-art hashing-based ANN solutions [18][19][20].

One example of such is additive quantization (AQ) [21]. It represents the original vectors as sums of vectors chosen from a limited set, the contents of which are learned from the data. AQ has been shown to achieve state-of-the-art search performance, but at the cost of computational complexity, which limits its applicability in practice.

Finding an accurate AQ representation for any given vector is called *encoding* and is known to be NP-hard, leading to the adoption of various heuristics, but even then it con-

stitutes the majority of AQ computational costs. Any vector to be added to the database needs to be encoded; the learning of AQ also requires numerous runs of encoding. Developing encoding algorithms that achieve the balance between complexity and accuracy is thus of major importance, as outlined by the AQ authors [21].

Several variants of AQ formulation have been proposed since, each taking a different approach to simplification of the encoding problem [22][23]. These approaches take form of additional constraints, resulting in some generality loss. As the representation becomes more limited, the search performance is decreased. However, the computational gains from the simpler encoding result in more practical solutions.

This thesis work aims to provide a novel encoding algorithm that does not rely on explicit constraints. Employing a previously unused heuristic, the goal is to achieve the compromise between the complexity and quality of encoding.

1.1 Thesis outline

The structure of the remainder of the thesis is described below.

Chapter 2 provides the problem formulations for both exact and approximate nearest neighbor search. Rough outline of the most common exact solutions follows. Finally, a very large family of hashing-based methods for approximate is presented. These approaches have been extensively researched, received a wide variety of practical applications and remained dominant in performance until the emergence of vector quantization family.

Chapter 3 presents the approximate search schemes based on vector quantization. Basic concepts common to many methods are presented first, followed by detailed descriptions of two most influential solutions – product quantization (PQ) and additive quantization (AQ). Several derivative methods are also briefly covered. Complexity and memory requirement estimates are provided for all the operations (if possible).

Chapter 4 explores the proposed encoding methods – pyramid encoding and residual pyramid encoding – from motivation to application and related costs. The influence and choice of initialization is discussed.

Chapter 5 covers the experimental evaluation of the proposed methods against the reference results of other recent approaches. Complexities are compared, as well as practical performance on image descriptor datasets in terms of quantization error and precision.

Chapter 6 contains the general conclusions about the work results. Further analysis of the proposed solutions is given, and several potential directions of future research are outlined.

2. NEAREST NEIGHBOR SEARCH

2.1 Problem formulation

Nearest neighbor (NN) search is a problem frequently encountered in many data-related fields and applications. It can be formalized as follows: Given N *database vectors* $X = \{x_1, x_2 \dots x_N\}$ in D-dimensional space (*search space*), a *query vector* q in the same space and a *distance measure* $dist(x, y)$, find a vector $NN(q) \in X$, such that NN is the closest to q:

$$NN(q) = \arg\min_{x \in X} dist(q, x) \tag{1}$$

The same formulation can be easily generalized to *k-NN search*, when, instead of a single nearest neighbor, a specific number k of vectors with smallest distances to q is returned. In this work only 1-NN search is considered, but all the methods and approaches listed can be trivially generalized to fit $k > 1$ scenario. Given a list of distances to database vectors, it is possible to utilize partial sorting techniques to retrieve k smallest values in $O(N + k \log k)$, where $k \log k$ can be disregarded if k is small enough [24].

Distance function in the above expression can be arbitrary, but while it is enough to use only a dissimilarity matrix, many methods make assumptions on specific distance types, relying on them for theoretical derivations or utilizing their properties for better performance. *Euclidean distance* is one of the most commonly used and well-researched distance metrics. For two D-dimensional vectors it is defined as follows:

$$dist_{EU}(x, y) = \sqrt{\sum_{d=1}^{D} (x_d - y_d)^2} \tag{2}$$

As the square root is a monotonously increasing function, its application does not change the comparison results between two non-negative values. Therefore, for neighbor search purposes, *squared Euclidean distance* is equivalent in practice, while saving some computation. Unless explicitly stated otherwise, any mention of a distance measure made within this work refers specifically to the squared Euclidean distance.

In many modern applications the exact nearest neighbor search is not practical [25]. This is mainly driven by the computational considerations, as many traditional search methods fail on data of sufficiently high dimensionality (see Section 2.2). Additionally, when performing a relevance search (information retrieval) in large databases (e.g. web search, document or image repositories) the exact solution is not strictly necessary, and any result which is sufficiently similar may be deemed acceptable. In these cases an

approximate nearest neighbor (ANN) search may be more suitable, giving up the exactness of the original NN problem to enable faster computation. Several ANN formulations exist; the most common one is *$(1+\varepsilon)$-approximate NN search***.** It requires finding a vector $\tilde{x}$ such that

$$dist(q,\tilde{x}) \leq (1+\varepsilon)dist(q,x_T) \tag{3}$$

where x_T is the true nearest neighbor and ε is a small positive real value. In other words, the distance from the query to $\tilde{x}$ should be no larger than the distance to the true nearest neighbor, scaled up by the factor of $1+\varepsilon$.

Because of non-exact nature of the ANN, the performance measure is required to assess how close the results are to the "ground truth". One commonly used measure is *recall@T***,** which is the probability that a true nearest neighbor is found within top T search results returned by a given method. The recall is typically estimated at pre-specified cutoff points (e.g. T = 1, 10, 100) of the approximate ranking. Alternatively, a recall curve (with T on the horizontal axis) is plotted to provide a visual representation of the method performance over many possible cutoff points.

2.2 Exact solutions

The straightforward solution of the exact NN problem is a so-called *linear search*. All the distances between q and the database vectors are calculated, while the lowest value calculated so far is retained. Since all the database vectors need to be considered, assuming that the distance function is linear in data dimensionality D, the linear search complexity is $O(ND)$. While acceptable for small datasets, the costs become prohibitive in many modern problems, specifically in Big Data environments.

Branch and bound approach has commonly been applied to a variety of optimization problems [2]. It reduces the total number of evaluated solutions by decomposing the search space in a manner such that the whole regions can be discarded at once. Search trees implement branch and bound paradigm for fast data searching and retrieval [1]. Numerous variants of trees have been proposed since, all of them sharing a single very desirable property – $O(\log N)$ complexity of search operations.

k-d tree is a classic search tree designed for multidimensional case [26]. It allows for both range searches and nearest neighbor queries, featuring logarithmic average complexity of insertions, deletions and retrievals (worst case is linear). The tree is constructed by recursively splitting the data space along dimensions; splitting points can be mean or median values of corresponding vector components. Nearest neighbors can then be retrieved by a procedure similar to depth-first search: the first leaf node reached is taken as a current best, and then other subspaces that can potentially contain closer points are inspected. Figure 1 shows the example of 2-d tree.

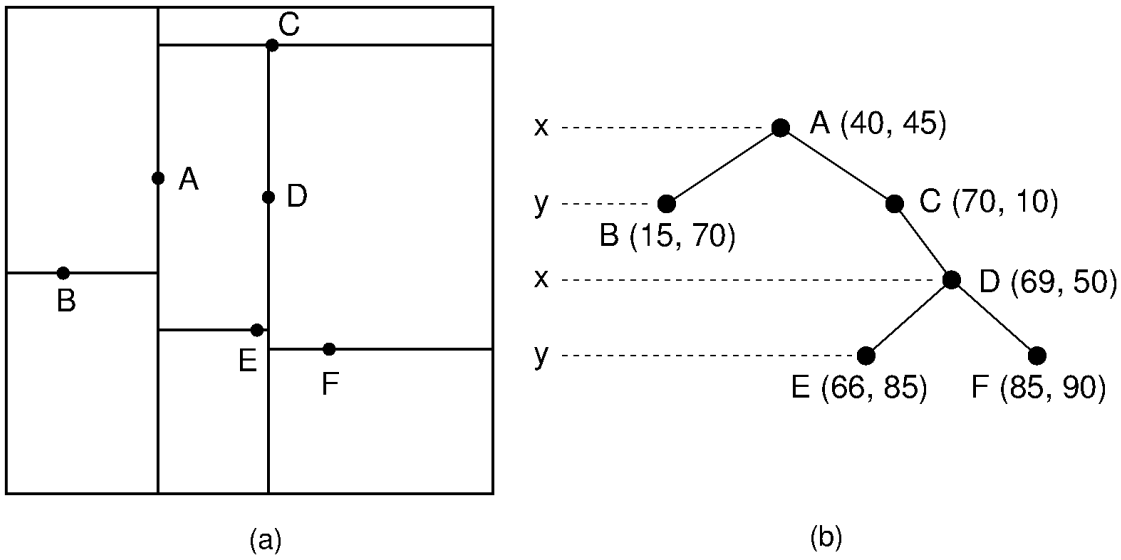

Figure 1. *Example of k-d tree for 2-dimensional data.* [1]

The approach taken by k-d trees to split the search space has significant drawbacks. Recursive partitioning leads to exponentially growing number of regions as dimensionality increases. The splitting points may be expensive to compute for large data sizes, and random sampling, which was introduced to mitigate this drawback, may result in suboptimal partitions. The k-d tree often fails to take advantage of possible internal data regularity, instead creating "dead" regions (containing no database points).

In an attempt to improve upon the k-d tree, many variants and modifications of it emerged over the years. *Balltrees* [27] represent a set hyperspheres instead of dimensional splits; these hyperspheres are not required to cover the data space completely and also allow regions to intersect or include each other. *Vantage point tree* (*vp-tree*) [28] further generalizes the problem to any metric space (instead of original Euclidean), performing splits based on similarity to a number of well-chosen *vantage points.* Figure 2 shows the k-d tree structure in comparison with vp-tree.

No matter what particular tree structure is used, some of their theoretical limitations cannot be overcome. Specifically, trees are not viable for high-dimensional data ($D > 10$). It has been demonstrated that in such scenarios they are reduced to exhaustive search with an additional overhead of maintaining the tree [5]. Query time therefore grows exponentially with dimensions; it is possible to make it polynomial, but only at the cost of making the preprocessing phase exponential instead [29]. Unfortunately, that makes trees unsuitable for many current applications; for example, the number of features in image or document retrieval easily reaches hundreds or thousands [30]. This shortcoming has become a large driving factor behind the recent large-scale research into approximate nearest neighbor techniques.

[1] Source: http://algoviz.org/OpenDSA/Books/Everything/html/KDtree.html

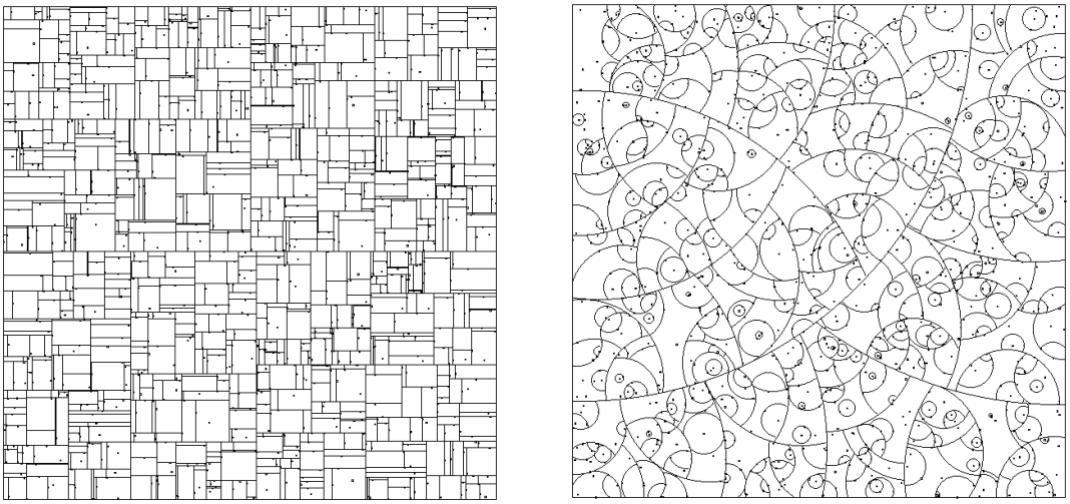

Figure 2. *k-d tree space decomposition (left) and vp-tree decomposition (right).* [28]

2.3 Hashing-based approximate search

In the context of ANN search, a *hash function* maps the original data space to another, significantly more limited space of *hash codes* [14]. The search problem becomes significantly easier to solve in the hash code space, allowing for a major computational gain. The approximate nature of the search stems from the imperfection of the hash function, as well as the inherent limitations of the new space.

Locality sensitive hashing (LSH) [25] is a family of hashing methods sharing the same design goal – similar items should be mapped to the same hash code ("bucket") with a high probability, while dissimilar items should be placed in different buckets. Naturally, if this requirement is to be fulfilled, different hash functions are required for different similarity measures. Constructing distance-specific hash functions with theoretically guaranteed bounds of performance is an ongoing topic of extensive research. Numerous solutions have been proposed in the recent years, presenting LSH function families for distances such as L_p (including e.g. Euclidean and Manhattan), cosine, Hamming, χ^2, as well as rank similarity, Jaccard coefficient for sets and general non-metric distances [14].

LSH family methods require the entire database to be hashed and stored in a code table. Then fast ANN search can be performed by hashing the query, locating the corresponding bucket and retrieving its contents (database vectors sharing the same code) for a linear search. LSH thus does not reduce the costs of a distance computation, but instead makes the search non-exhaustive, effectively performing randomized space partitioning not unlike the previously considered tree structures.

Figure 3 visually illustrates the concept. The red circle denotes the query, the hash code of which is "0110". Only those database vectors that share the same code will be re-

trieved for distance computation and comparison. The approximate nature of the representation is also apparent from the figure, as the immediate neighborhood of the query was not hashed into the same bucket, leading to loss of search accuracy.

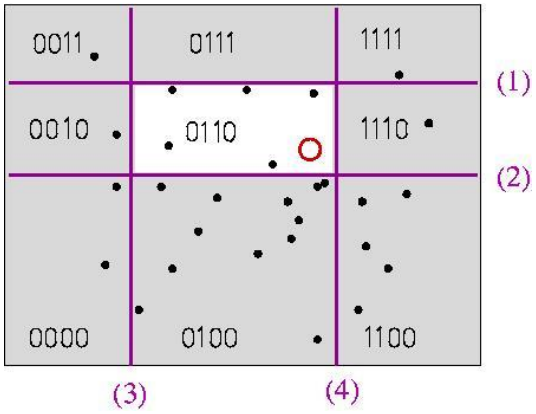

Figure 3. *Visual example of locality sensitive hashing.*[2]

To improve the search performance, it is common to use several hash functions simultaneously [14]. For storage purposes the resulting codes are concatenated into a single vector, but during the retrieval the query is independently hashed with each function, and every corresponding bucket contributes its contents. The number of vectors for distance calculation thus increases, but so does the accuracy.

One of the common LSH schemes for Euclidean distance is based on so-called 2-stable distributions [31]. An example of such a distribution is a Gaussian. The hash function is then given as

$$h(x) = \left\lfloor \frac{w^T x + b}{r} \right\rfloor, \tag{4}$$

where x is a D-dimensional database vector, w is a random D-dimensional vector sampled from the Gaussian distribution of zero mean and unit variance, r is a positive real number and b is an offset value, randomly chosen from a range $[0, r]$. The database vector is thus mapped to a single value via random projection and shifting.

Hash functions in the LSH family of methods are completely data-independent, making applications easier, but sacrificing performance. To alleviate this, several *learning-to-hash* approaches have been proposed [14]. They aim to derive and/or tune an efficient hash function from the specific data with respect to some optimization criterion, not unlike the training phase of machine learning models. A common secondary criterion is the maximization of the coding efficiency by maintaining bit balance and bit independence. Bit balance implies that for each code bit the values 0 and 1 are approximately equally probable. Bit independence is hard to attain and is in practice relaxed to decor-

[2] Source: http://ttic.uchicago.edu/~gregory/pose/psh.html

relation; the purpose of both is minimizing the code redundancy. Together these requirements allow for the shortest codes possible given a particular data.

One influential example of an early data-specific hashing algorithm is *spectral hashing* [32], which presents the similarities in a graph form and then infers binary codes as the balanced partitioning of the graph. Since the problem is NP-hard, it is solved approximately by eigen-analysis of the graph Laplacian. To generalize the solution to the data not used for training, it is assumed that the input vectors are generated by the uniform distribution. Since this assumption is usually violated in practice, the performance of spectral hashing is degraded; the algorithm, however, provides important theoretical insights and has inspired many learning-to-hash approaches since.

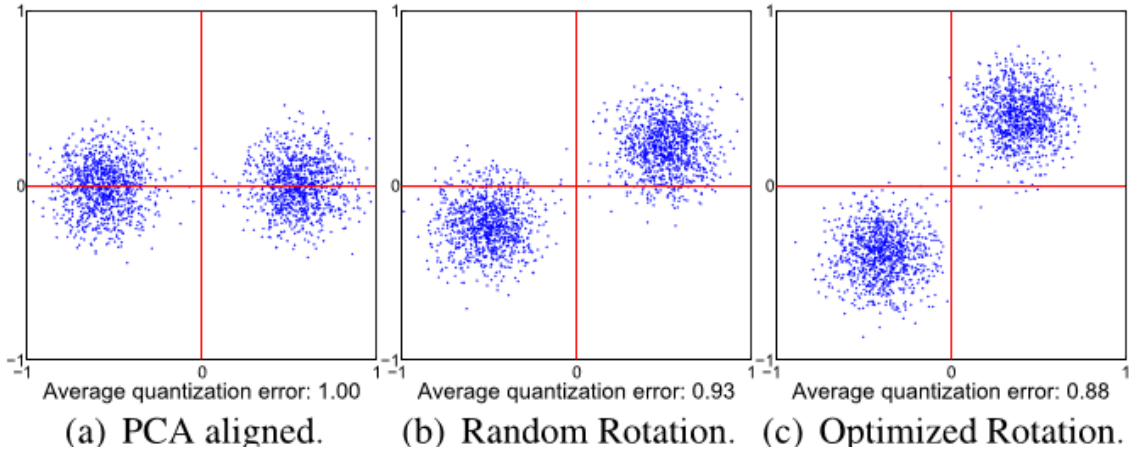

Figure 4. *ITQ on preprocessed normalized data.* [33]

Iterative quantization (ITQ) [33] is an efficient learning-to-hash algorithm, which assigns every database vector to a closest (in Euclidean terms) vertex of an M-dimensional hypercube ($M \leq D$). Every vertex can then be uniquely indexed by M bits, resulting in binary hash codes. First the data dimensionality is reduced to M via principal component analysis (PCA). Then, to improve the hashing, rotation is applied to the preprocessed data, for the purpose of balancing the variances along each principal direction. This corresponds to previously described bit balance requirement. Even the random rotation improves the hashing, but ITQ instead opts for an optimized rotation, given the PCA-aligned data. This transformation is found by minimizing the Euclidean distance between each database vector and the hypercube vertex it is assigned to. The example can be seen on Figure 4. If no rotation is applied (Figure 4a), similar data points (contained in the same cluster) will be assigned to different hypercube vertices, resulting in different hash codes. Random rotation (Figure 4b) provides an improvement, but optimized rotation (Figure 4c) attains the best solution.

ITQ, as its name implies, is learned by an iterative procedure, alternating between assignment of data points to vertices and optimizing the rotation. These two steps are repeated until no further changes in hash codes occur.

3. VECTOR QUANTIZATION FOR APPROXIMATE SEARCH

3.1 General concepts

Vector quantization (VQ) [16] is a technique that represents a given set of N D-dimensional vectors with another set of K *centroids* of the same dimensionality ($K < N$). The set of centroids C is called a *codebook*, while centroids themselves are alternatively called *codevectors*. Each vector from the original set is represented by one and only one codevector. Needless to say, VQ representation of the data is lossy. The quantization loss is typically measured by mean squared error (MSE) between the data vectors and their reproductions:

$$E = \frac{1}{N}\sum_{i=1}^{N}\|q(x) - x\|_2^2, \tag{5}$$

where x is the data vector and $q(x)$ is the corresponding codevector.

Any optimal (having minimal quantization loss) VQ quantizer is subject to two necessary optimality conditions, known as *Lloyd conditions* [17]. First condition states that each vector must be assigned to a centroid which is the closest in Euclidean distance terms:

$$q(x) = \arg\min_{c_i \in C} \|c_i - x\|_2^2. \tag{6}$$

Second condition limits the position of each centroid to the mean value of all the vectors it represents:

$$c_i = \frac{1}{n_i}\sum_{j=1}^{n_i} x_j\,,\ s.t.\ q(x_j) = c_i \tag{7}$$

These conditions are greatly reminiscent of well-known k-means clustering. Indeed, a common way to construct a vector quantizer is with a *Lloyd's algorithm*:

1. Randomly initialize centroid positions.
2. Repeat for a predetermined number of iterations:
    a. Assign every data vector to the nearest centroid (6).
    b. Replace every centroid with the mean of vectors assigned to it (7).

The total computational complexity of Lloyd's algorithm is $O(LNKD)$, where L is the number of iterations. While Lloyd's algorithm is intuitive, simple and widely used, it only converges to a locally optimal solution. The results may vary wildly with different initializations, and some starting points may lead to very poor representations.

A set of vectors assigned to a particular centroid is known as *Voronoi cell* or just a *cell*. Any vector quantizer therefore defines a space partitioning – a *Voronoi tessellation*, with each cell defining a separate subspace. An example of such is shown on Figure 5.

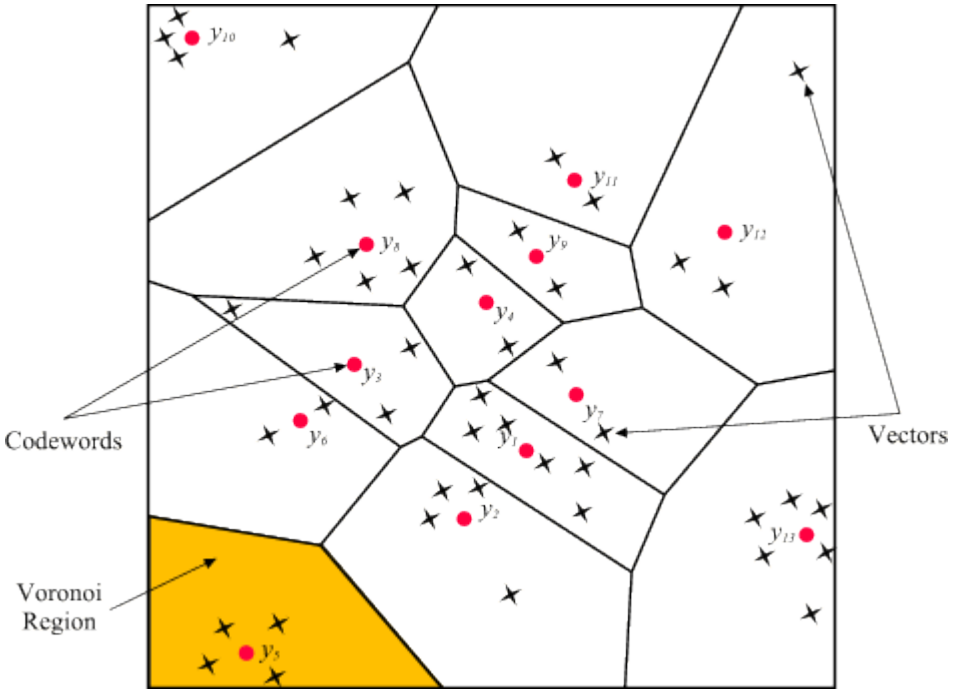

Figure 5. *An example of a vector quantizer and corresponding Voronoi tesselation.*[3]

Evidently, the Voronoi tessellation can be used as a foundation for non-exhaustive distance calculation. One strong advantage of such an approach, when compared to hashing, would be the fact that the original data space is preserved, and the Euclidean distances remain meaningful instead of being approximated with Hamming distances. Better preservation of pairwise vector dissimilarities, in turn, leads to better search performance.

Since vector quantization does not change the distance function, there are two possible approaches to the nearest neighbor search [18]. If *symmetric distance computation* (SDC) is used, the query vector is quantized to the nearest centroid, and then the distances from that centroid to all the others are estimated. In case of *asymmetric distance computation* (ADC), the distances between the query and all the centroids are calculated directly. Both scenarios are shown on Figure 6.

[3] Source: http://www.mqasem.net/vectorquantization/vq.html

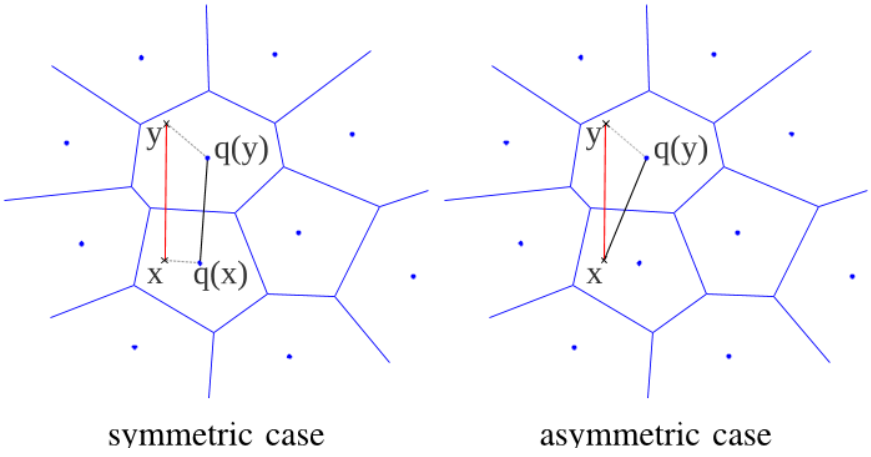

Figure 6. *Distance computation with vector quantization: symmetric (left) and asymmetric (right).* [18]

Since centroid positions are always known in advance when the search is performed, it is possible to precompute all the pairwise distances between centroids and store them. This seemingly makes SDC very quick, but to quantize the query, one needs to calculate the distance from it to every centroid, which is the exact same process as in ADC. As a result, the differences in the calculation speed between the two are minimal. However, the quantization of the query vector introduces additional distortion. It is thus reasonable to conclude that ADC is superior to SDC and should be preferred. In fact, availability of ADC is a direct consequence of space preservation and can be considered an additional advantage of vector quantization over hashing-based approaches.

Despite the aforementioned benefits, traditional vector quantization without any changes is not a practical solution for approximate nearest neighbor search, as it too suffers from the "curse of dimensionality" [18][34]. If the number of codevectors (centroids) is K, the code (index) of each database vector has a length of $\log_2 K$ bits. This is not nearly enough for discrimination; for example, if a vector of dimensionality 960 (e.g. global color GIST descriptor [35]) would be represented by a 960-bit code (1 bit per dimension), the corresponding vector quantizer would have to have 2^{960} centroids. Evidently, it is impossible to even store a codebook of that size in a computer memory, let alone applying any search operations on it. A different approach is necessary to take advantage of VQ properties. A number of such approaches have emerged, proceeding to outperform hashing-based techniques by a large margin [18][19][20].

3.2 Product quantization (PQ)

Product quantization (PQ) [18] is one of the most important methods in the vector quantization family. To allow for more powerful representations with small number of codevectors, PQ splits the data space into M subspaces, each of D/M dimensions. Vector quantization (learned with Lloyd's algorithm) is then separately applied on each

subspace, resulting in M codebooks. Every database vector can be subsequently reconstructed by concatenating M corresponding codevectors. Assuming that each codebook has K codevectors, the total number of possible representations is K^M. Any quantized database vector is stored as a sequence of M codes, indexing into K elements each, resulting in a total code length of $M \log_2 K$ bits. The amount of memory required for codebook storage (in terms of scalar values) is $M \cdot K \cdot \frac{D}{M} = KD$, which is small in practice, as $K \ll N$.

Figure 7 shows an example of a product quantizer applied to 128-dimensional vector. It is typical to use parameter values which are powers of 2, as indices are stored in a binary computer memory. In this case $M = 8$ codebooks are used, and each 16-dimensional subspace is quantized to $K = 256$ centroids (codevectors). A single subspace can then be indexed with an 8-bit ($\log_2 256 = 8$) code. The whole vector is subsequently stored as a concatenation of sixteen 8-bit codes for a total of 64 bits.

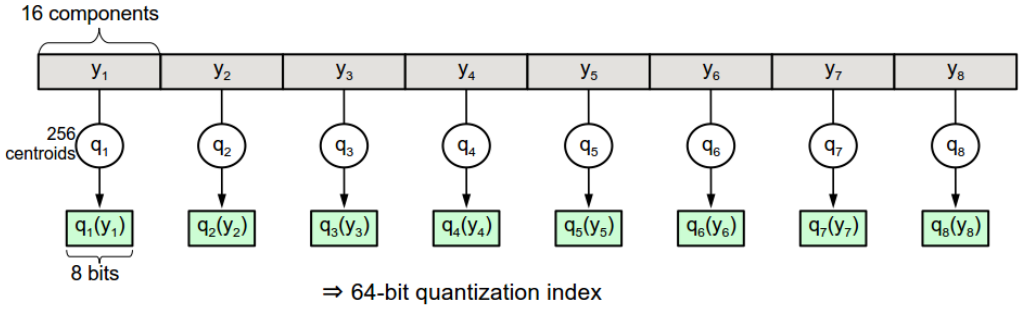

Figure 7. *Product quantizer for 128-dimensional vectors,* $M = 4,\ K = 256$.[4]

3.2.1 PQ distance estimation

Being a vector quantization-based approach, PQ allows both symmetric and asymmetric distance estimation. The previous conclusion in regards to the superiority of ADC (see Chapter 3.1) applies here; its advantage in ANN search was also shown experimentally by the authors [18]. For this reason only ADC is considered for search purposes in the following section.

Assume that the database vectors are quantized with codebooks $C_i, i = 1..M$. If every codevector is padded with zeros to the original number of dimensions D, a reconstruction vector $\tilde{x}$ can be represented as a simple sum: $\tilde{x} = \sum_i^M c_i$. Note that the padded codevectors are orthogonal to each other as long as they come from different codebooks. Then the squared Euclidean distance between $\tilde{x}$ and a query vector q decomposes to the sum of distances over the individual subspaces:

$$\|q - \tilde{x}\|_2^2 = \sum_i^M \|q - c_i\|_2^2 - (M-1)\|q\|_2^2 \qquad (8)$$

[4] Source: http://lear.inrialpes.fr/pubs/2010/JDSP10/jegou_compactimagerepresentation_slides.pdf

The second component of (8) does not depend on $\tilde{x}$; as a result, it does not affect the relative ranking between different database vectors and can be ignored during search, unless the estimated values of distances are required. Taking into account the codevector structure, the total number of operations required to compute (8) is $O(M \cdot \frac{D}{M}) = O(D)$ – the same as if the distance was computed without quantization. The benefit of PQ is in the fact that the same set of codevectors is used to represent the whole original database.

It is possible to compute pairwise distances between a given query vector and all the codevectors (from all codebooks), storing them into a table. This would require $M \cdot K \cdot \frac{D}{M} = KD$ operations, taking subspaces into account. Then the value of $\|q - c_i\|_2^2$ can be found in constant time via a table lookup, meaning that $\|q - \tilde{x}\|_2^2$ can be calculated in M lookups and M summations. Estimating the distances from the given query to a quantized set of N vectors would require $O(KD + NM)$ total operations. This is significantly smaller than the linear search of $O(ND)$. Since the dimensionality and number of vectors are decoupled, product quantizer allows for much better scaling on both.

One drawback of such an approach is the exhaustive nature of the search. When N is very large, even the product quantizer costs may become excessive. *Inverted indexing* is one of the possible solutions for this problem [18]. Simple vector quantization is applied on the data, with a number of centroids $k' < N \ll K^M$. The value of k' typically ranges anywhere from 10^3 to 10^6. Then for each data point the residual is calculated between it and the centroid of its cell:

$$r(x) = x - q(x) \tag{9}$$

These residuals are subsequently quantized with PQ, and both representations are retained. When the query is processed, the centroids of the coarse quantizer (VQ) are exhaustively searched first, requiring $O(k'D)$ operations. When the nearest centroid is located, query residual is computed and the normal PQ distance estimation is performed, but only in a single Voronoi cell, which contains some subset of the database. This allows for significant search speedup at the cost of additional memory requirements [18]. Although it's possible to learn a separate product quantizer in each Voronoi cell, this would be prohibitive for large k', meaning that a single set of PQ codebooks is used for all the residuals. Additional memory costs are thus $O(k'D)$ to store the coarse quantizer and $O(N \log_2 k')$ for all the corresponding indices.

It is common in practice that the database vector and its true nearest neighbor would be assigned to the separate centroids of the coarse quantizer. This means that the procedure outlined above could not possibly locate the true neighbor, as it is not included in the data subset to be searched. To alleviate this, a multiple assignment strategy is proposed [18][36]. During the coarse quantization step, each database vector is assigned not to a

single closest centroid, but to w nearest centroids. All of their contents are consequently retrieved for exhaustive searching. The PQ authors have concluded that inverse indexing with multiple assignment provides a major speedup for large databases and even improves the search performance in some cases [18].

3.2.2 Learning the product quantizer

To encode a data vector with PQ (given a set of codebooks), it is necessary to locate the closest centroid (in Euclidean terms). PQ centroids are generated by the Cartesian product of the codebooks, so the total number of options is, as mentioned earlier, K^M. This is too large for an exhaustive search. Fortunately, subspace orthogonality can be exploited here, same as for the query distance estimation. The distance between a given vector and a PQ centroid decomposes into the sum of distances to codevectors within individual subspaces, yielding the exact same result as (8). The nearest neighbor search in K^M vectors is thus replaced with M nearest neighbor searches in K vectors each; the indices of the located codevectors are then concatenated to obtain the result. Computational complexity of encoding a single vector becomes $O\left(M \cdot K \cdot \frac{D}{M}\right) = O(KD)$. It is important to note that encoding costs do not depend on the number of codebooks M.

Before PQ can be applied on a dataset, it is necessary to obtain a good set of codebooks. Because the codebooks are data-specific, they need to be *learned* in a process known as *training*. Product quantization inherits its training scheme directly from vector quantization, adapting a simple Lloyd algorithm for iterative adaptation. To save computation time, it is common to not use the whole data for training, opting instead for a representative subset.

As discussed earlier, Lloyd's algorithm alternates between centroid assignment and centroid adjustment. The former corresponds to encoding the training data, which was already described. The latter involves adapting the codebooks to minimize the quantization error, given a particular encoding of the data. As PQ is equivalent to VQ in each individual subspace, optimal codevectors are generated by averaging their corresponding Voronoi cells. PQ training is thus completely equivalent to running M k-means clustering algorithms in reduced dimensionality spaces. With L training iterations and N training vectors, the total complexity of learning a set of codebooks is $O(NLKD)$.

To start the training, it is necessary to define a starting point – either an initial encoding or an initial set of codebooks. Although it's possible to use random (e.g. uniform) codes for this purpose, the common approach is to generate the codevectors by sampling from the corresponding dimensions of training data [18].

3.2.3 Optimized product quantization (OPQ)

Optimized product quantization (OPQ) [19], also known as *Cartesian k-means (CKM)* [37], is a variant of product quantization that makes an attempt to optimize the allocations of dimensions to subspaces. In the original PQ paper [18] it was noted that the search performance varies greatly based on the contents (semantics) of the data. Product quantization considers all the subspaces to be equal in terms of information content, which is quite likely to be incorrect for the real sets of vectors. In fact, the choice of dimensions for a particular subspace has been shown to have a major effect on the quantization performance [18]. Since the domain knowledge is not always available, the quantization algorithm can benefit from an internal subspace optimization.

Rotation is a linear transformation that preserves vector norms and pairwise Euclidean distances; for any orthogonal $D \times D$ matrix R and D-dimensional vectors x and y the following expression holds:

$$\|x - y\|_2^2 = \|Rx - Ry\|_2^2. \tag{10}$$

Due to (10) any centroid assignments (codes) of a vector quantizer with codebook C on dataset X remain valid, if both codevectors and data vectors are transformed with the same matrix R. This property naturally generalizes to PQ. The benefit of rotation lies in the fact that it can represent any reordering of the vector dimensions [19]. Optimizing rotation of PQ quantizer is thus equivalent to finding better allocation of dimensions to subspaces. Random rotation has been explored and found to improve the quantization error and search performance [38]. Further gains can be expected if the matrix R is fine-tuned with respect to the data.

OPQ differs from PQ in that it learns not only codebooks and codes, but also an orthogonal transformation matrix R. Any data to be encoded is first rotated via multiplication by R, and then the product quantizer is applied. Computational complexity of encoding a single vector thus increases by the multiplication cost, resulting in a total estimate of $O(KD + D^2)$.

Two formulations of OPQ exist [19]. Parametric formulation is derived from the assumption that the data is generated by Gaussian distribution. If the assumption holds, the solution is provably optimal and achieves the theoretical lower bound of quantization error, which is derived to be

$$E \geq \frac{D}{M} K^{-\frac{2M}{D}} \sum_{m=1}^{M} \left|\hat{\Sigma}_m\right|^{\frac{M}{D}}, \tag{11}$$

where $\hat{\Sigma}_m$ is the covariance submatrix of m-th subspace of rotated data Rx. Further theoretical analysis shows that for a parametric solution to achieve optimality, two conditions must hold: *subspace independence* and *balance of subspace variance*. These con-

clusions exactly correspond to the ones drawn in ITQ – a hashing-based method described earlier.

To construct a matrix R satisfying the above requirements for subspaces, a simple greedy algorithm – *eigenvalue allocation* – was proposed [19]. The database is first processed with PCA, which takes $O(ND^2 + D^3)$ operations. The eigenvalues are then sorted in a descending order and sequentially allocated to M bins of capacity D/M, such that the product of values inside each bin would be roughly equal (thus balancing the variance). When the eigenvectors are reordered according to allocation of their respective eigenvalues, the matrix R is formed. Then the database is rotated and PQ is applied on the result. Total training complexity of parametric OPQ is thus $O(ND^2 + D^3 + NLKD)$.

Naturally, the Gaussian assumption rarely holds in practice. Nonparametric OPQ does not construct a single rotation matrix, but instead optimizes it during the training phase [19]. At the beginning of each iteration the data is rotated with the current estimate of R. Then the codebook adaptation and encoding are performed on the rotated data, with no differences from PQ. Finally, the current estimate of R is updated so as to minimize the quantization error criterion:

$$\min_R \|RX - Y\|_F^2, \tag{12}$$

where X and Y are matrices composed of original (not rotated) data vectors and their reconstructions, respectively, and $\|\cdot\|_F$ is the Frobenius norm: $\|A\|_F = \sqrt{trace(A^*A)}$. To solve this optimization problem, first the singular value decomposition (SVD) of XY^T is calculated: $XY^T = USV^T$. The matrix R is then calculated as follows:

$$R = VU^T. \tag{13}$$

The complexities of this procedure are as follows: XY^T multiplication is $O(ND^2)$, SVD is $O(D^3)$, and VU^T multiplication is also $O(D^3)$. Since rotation is adapted in each iteration, the total cost of non-parametric OPQ training becomes $O(L(NKD + ND^2 + D^3))$.

Nonparametric OPQ typically performs better on real datasets, as it does not rely on Gaussian assumption. OPQ training can be initialized similarly to PQ, with an identity matrix as a first estimate of R. Better results are attained if parametric solution is used for initialization, although that increases the amount of computation required [19].

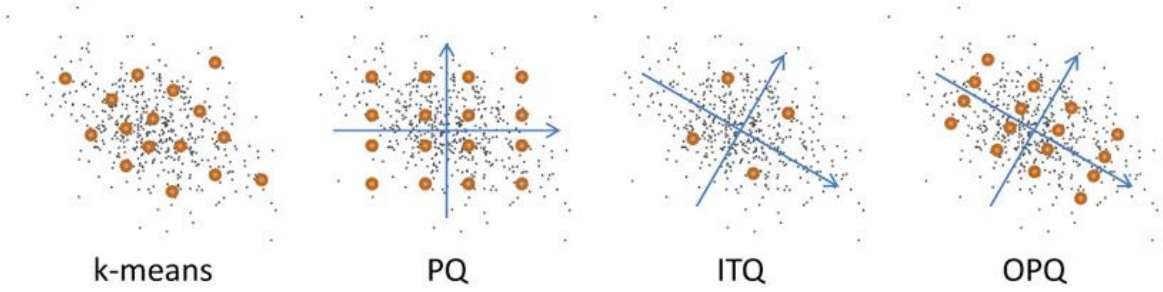

Figure 8. *Visualization of different quantizers trained on artificial data.*[5]

Figure 8 shows the centroids generated by k-means (VQ), PQ, ITQ (Chapter 2.3) and OPQ on artificial two-dimensional data. Adaptive procedure for subspace allocation allows OPQ to reliably outperform PQ and numerous hashing-based methods [19]. OPQ requires $O(KD)$ memory to store the codebooks (same as PQ) and $O(D^2)$ memory for the rotation matrix.

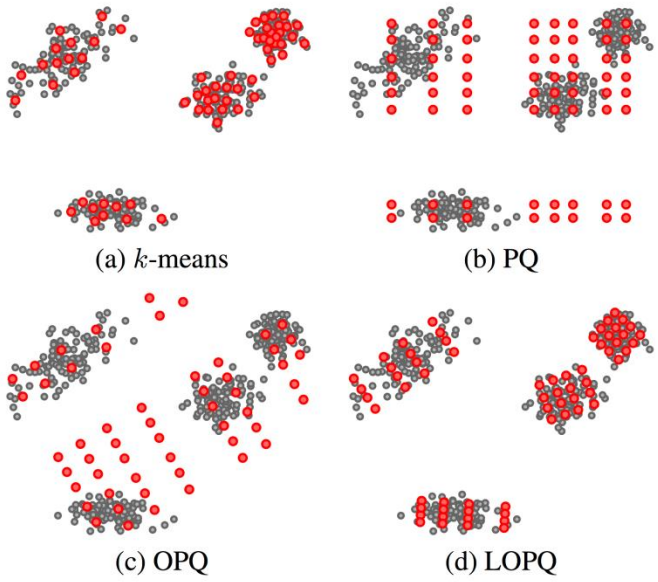

Figure 9. *Different quantizers on artificial data with well-separated clusters* [39]

A further derivate of optimized product quantization was recently proposed – *locally optimized product quantization* (LOPQ) [39]. The idea behind the approach is that optimizing separate rotation matrices for different regions of data space might lead to better performance compared to a single global transformation (see Figure 9). To achieve this, LOPQ applies standard k-means on the data and then trains a separate optimized product quantizer on each cluster. To index the vector on both hierarchical levels, $\log_2 K' + M \log_2 K$ bits are required, where K' is the number of centroids of k-means. For the sake of simplicity it is typically assumed that $K' = K$, as running too many sep-

[5] Source: http://kaiminghe.com/cvpr13/index.html

arate OPQ algorithms will increase the computational costs dramatically. Assume that the top-level quantizer is learned over L' iterations, and the data is roughly equally distributed between k-means clusters. Then it takes $O(L'KND)$ operations to run global k-means, followed by the costs of training K OPQ instances on N/K data points each, which is $O\big(KL(ND + ND^2 + D^3)\big)$. Adding the two together and simplifying the result leads to the following LOPQ training complexity: $O(L'NKD + LK(ND^2 + D^3))$.

In addition to the training overhead, other operations with LOPQ naturally have higher computational costs. K individual OPQ quantizers need to be stored, resulting in memory cost of $O(KD[K + D])$. Query processing requires searching the top level first, which takes extra KD operations, but the following OPQ distance estimation is non-exhaustive and takes $D^2 + KD + N'M$ operations, where $N' < N$. Finally, a vector encoding cost also increases by KD operations due to the need of locating the closest top-level centroid.

LOPQ requires noticeably more complex training, but its encoding and distance calculation overheads, as well as increase in code length, are considered to be well justified by the improvement in quantization error and search performance [39].

3.3 Additive quantization (AQ)

Additive quantization (AQ) generalizes the methods described earlier by relaxing codebook constraints [21]. The quantizer, similarly to PQ or OPQ, consists of M codebooks, each containing K codevectors. However, no subspace decomposition is performed, and all the codevectors share the same dimensionality D as the original data. Instead of concatenation, M indexed codevectors are instead added together to obtain a reconstruction (see Figure 10).

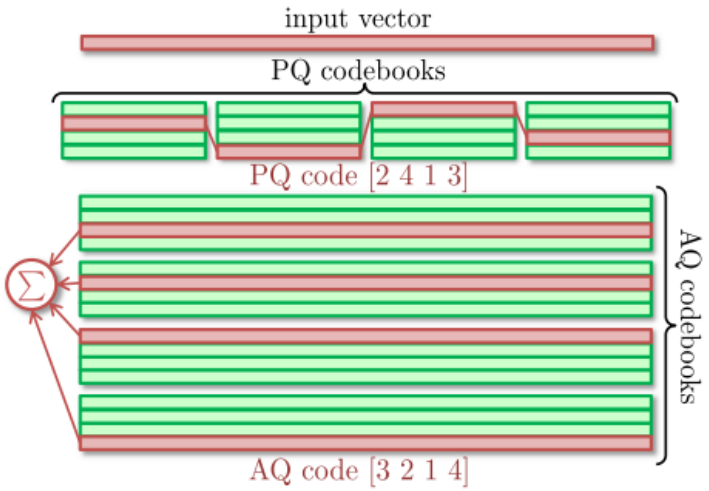

Figure 10. *Vector representations in PQ and in AQ.* [21]

The representation codelength is still $M \log_2 K$, just as in the methods of PQ family. Memory requirements for codebook storage are naturally higher, due to the increased codevector dimensionality, and constitute an equivalent of MKD values – M times larger when compared to PQ or OPQ. Since the value of M is rarely above 16, this extra cost is deemed negligible for sufficiently large datasets. Any set of PQ or OPQ codebooks may be converted to a set of AQ codebooks by appropriate padding of all codevectors with zeroes.

Elimination of subspace decomposition means the omission of codebook orthogonality constraint. This, in turn, leads to potentially richer representation, and AQ indeed has outperformed older methods significantly [21]; however, this formulation comes with the price of much higher encoding and training complexity, as simple k-means procedure is no longer applicable.

3.3.1 AQ distance estimation

The squared Euclidean distance from query q to a reconstructed vector $\tilde{x} = \sum_{i=1}^{M} c_i$ can be represented with the following expression:

$$dist^2(q, \tilde{x}) = \sum_{i=1}^{M} \|q - c_i\|_2^2 - (M-1)\|q\|_2^2 + \sum_{i=1}^{M} \sum_{j=1, i \neq j}^{M} \langle c_i, c_j \rangle \qquad (14)$$

When compared to (8), AQ distance estimate has an additional component – the sum of dot products between all the included codevectors. This is due to codebooks no longer being orthogonal. Since this component does not depend on q, it can be precomputed and reused for different queries. One approach is to construct a set of lookup tables, containing dot products between all the possible pairs of codevectors. In this case $M^2/2$ extra lookups will be required for every search, in addition to $O(M^2K^2)$ memory. Another solution would be to calculate the sum in (14) for each encoded vector in the database and store it as set of scalars. To reduce the memory overhead, these scalar values can be quantized and appended to the AQ code of each data point; then the dot products in (14) can be found with a single lookup. Although this operation further distorts the distances and degrades the search performance, experiments have shown that the detrimental effects are minimal and may well be justified by the faster search [21].

The first two terms of (14) are identical to their PQ equivalents. The only difference is that full-length codebooks result in more expensive lookup table calculations – $O(MKD)$ instead of $O(KD)$. However, since M is relatively small in practice (rarely taking a value above $M = 16$), this overhead is considered acceptable, especially for large databases [21]. The total search cost for a single query therefore becomes $O(MKD + MN)$, or $O(MKD + M^2N)$ if the dot product table lookup is used.

3.3.2 Learning the additive quantizer

An additive quantizer is trained iteratively, via alternating between codebook adaptation and encoding. Unfortunately, the distance estimate of (14) is more complex than (8), so a simple problem decomposition is not possible; general approaches for both of the steps need to be devised instead.

Given fixed codes, an optimal set of codebooks can be found by solving the following least-squares problem:

$$\min_{C_{mk}} \sum_n^N \left\| x_n - \sum_m^M \sum_k^K a_{nmk} C_{m,k} \right\|_2^2, \tag{15}$$

where $C_{m,k}$ is the k-th codevector of m-th codebook and a_{nmk} is a binary indicator variable, taking value 1 if a database vector x_n is assigned to codevector C_{mk} and zero otherwise. The corresponding system of linear equations is overdetermined, with the corresponding matrix having N rows and MKD columns. Equivalently, D overdetermined linear systems of N equations and MK variables each can be solved, as problem (15) decomposes along the dimensions [21]:

$$\begin{cases} \sum_m^M \sum_k^K a_{1mk} C_{m,k1} = x_{11} \\ \sum_m^M \sum_k^K a_{2mk} C_{m,k1} = x_{21} \\ \dots \\ \sum_m^M \sum_k^K a_{Nmk} C_{m,k1} = x_{N1} \end{cases}, \quad \dots, \quad \begin{cases} \sum_m^M \sum_k^K a_{1mk} C_{m,kD} = x_{1D} \\ \sum_m^M \sum_k^K a_{2mk} C_{m,kD} = x_{2D} \\ \dots \\ \sum_m^M \sum_k^K a_{Nmk} C_{m,kD} = x_{ND} \end{cases} \tag{16}$$

Fortunately, since only M out of MK coefficients in each equation are nonzero, the coefficient matrix can be efficiently stored in sparse format (with density $1/K$), allowing for special solvers to be used. In addition, the coefficient matrix is exactly the same for all the systems (only the right-hand constant terms x_{id} change), so it can be reused for all the solutions. These factors lead the AQ authors to disregard the computational costs of codebook adaptation during training, as the complexity is dominated by the encoding step [21].

The problem of finding the optimal AQ representation of a given vector is equivalent to inference on a fully connected pairwise Markov Random Field (MRF), which is NP-hard [21][40]. One of the simplest algorithms for this problem is *Iterated Conditional Modes* (ICM) [41]. In the context of AQ encoding ICM proceeds as follows:

1. Given a database vector x, set iteration counter $l = 1$, current reconstruction $\tilde{x} = 0$ and codevector assignments $i_m = 0, m = 1 \dots M$.
2. For all the codebooks $m = 1 \dots M$ do:
    a. If $i_m \neq 0$, subtract the currently assigned codevector from the reconstruction: $\tilde{x} = \tilde{x} - C_{m,i_m}$.
    b. Calculate the current residual $r = x - \tilde{x}$.

c. Find the nearest neighbor to the residual amongst the codevectors of the current codebook and take it as an assignment:

$$i_m = \arg\min \left\| r - C_{m,i_m} \right\|_2^2 \tag{17}$$

d. Modify the reconstruction accordingly: $\tilde{x} = \tilde{x} + C_{m,i_m}$ and proceed to the next codebook.

3. Increase the iteration counter $l = l + 1$, if pre-specified maximal number of iterations L was reached, stop, otherwise go to step 2.

As can be seen from the above, ICM considers one codebook at a time and tries to improve the quantization error, while keeping other $M - 1$ assignments fixed. This solution has a complexity of $O(LMKD)$ for encoding a single vector; unfortunately, as it never considers interaction between codebooks, its greediness leads to poor practical performance. Other MRF-specific methods were also considered by the AQ authors, but failed to achieve suitable results, prompting them to suggest their own approach [21].

Beam Search is a heuristic method of AQ encoding, based on 2-step optimization. Beam Search concatenates all the codebooks thus considering more options at a time. Negative effects of the greedy encoding manifest as globally poor codevector choices, driven by the local optimality. To alleviate this, Beam Search does not improve a single solution throughout the algorithm, but instead keeps H candidate solutions, the best of which is chosen after the procedure is over. *Search depth H* is a parameter set before the encoding. Figure 11 shows an illustration of the first step of Beam Search.

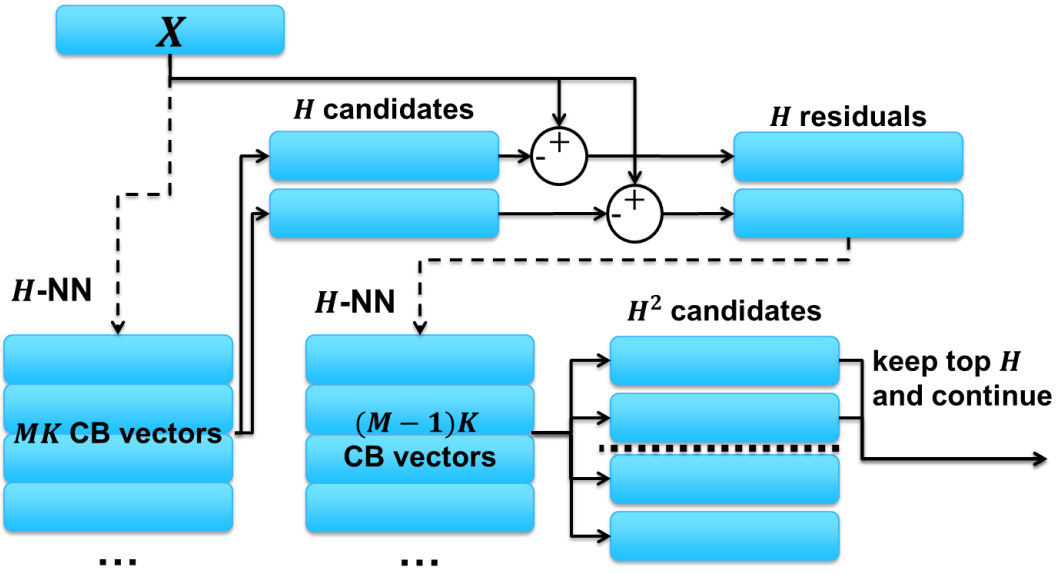

Figure 11. *Visual representation of the first iteration of Beam Search encoding.*

The entire Beam Search encoding sequence is given below:

1. Given a database vector x and search depth H, set codevector assignments $i_m = 0, m = 1 \dots M$.
2. Find H nearest neighbors to x from all the MK codevectors and store them, as well as corresponding codebook indices, as candidate solutions:
   $I = \{\{\bar{m}_1, i_{\bar{m}_1}\} \quad \{\bar{m}_2, i_{\bar{m}_2}\} \quad \dots \quad \{\bar{m}_H, i_{\bar{m}_H}\}\}$.
   Here $\bar{m}_j$ is a set of codebook indices and $i_{\bar{m}_j}$ is a set of corresponding codevector assignments.
3. For every candidate solution I_h, $h = 1 \dots H$ do:
   a. Calculate the solution residual:
      $r = x - \sum_m C_{m,i_m}, m \in \bar{m}_h$.
   b. Find H nearest neighbors i_{m^*} to r from $C' = \bigcup C_m, m \notin \bar{m}_h$, i.e. from all codevectors whose codebooks are not yet in the solution.
   c. Generate H new solutions by appending the indices found in the previous step to the current solution:
      $\bar{m}_{hj} = \bar{m}_h \cap m_j^*, \; i_{\bar{m}_{hj}} = i_{\bar{m}_h} \cap i_{m_j^*}, \; j = 1 \dots H$.
   d. Replace the current solution with H newly generated ones:
      $I = I \setminus \{\bar{m}_h, i_{\bar{m}_h}\} \cap \{\bar{m}_{h1}, i_{\bar{m}_{h1}}\} \cap \dots \cap \{\bar{m}_{hH}, i_{\bar{m}_{hH}}\}$
4. I is now a list of H^2 solutions. Sort the list by the quantization error and keep H best solutions.
5. If every solution in I contains M codevectors, take the solution with the lowest quantization error as a final answer and stop. Otherwise go to step 3.

Two-step optimization allows Beam Search encoding to find significantly better representations. The major drawback lies in the complexity of the procedure. The algorithm described above has the complexity of $O(M^3KHD)$, which is the highest amongst the current quantization-based approaches. Some improvement can be attained by avoiding the distance calculations within the encoding process, replacing them with table lookups. The table can be constructed in $O(MKD)$, resulting in a total complexity of $O(M^3KH + MKD)$. In this case the data dimensionality has a lesser effect on encoding, but the cost remains very high. It is particularly pronounced during the training phase, where the dataset has to be repeatedly re-encoded over several iterations.

AQ with Beam Search encoding has demonstrated superior results in both search and quantization error reduction, improving upon the majority of other methods [21]. However, the prohibitive complexity hinders the practicality of AQ applications. Training time is typically given less consideration in ANN search systems, as it is performed in offline mode and can be assumed to take advantage of the full computational power available. Encoding new database vectors, on the other hand, is a relatively common task and can be considered time-sensitive.

AQ authors suggest a simple approach to slightly reduce the computational costs of AQ while sacrificing some representation power. As AQ scales cubically with the number of codebooks, it may be viable to perform explicit subspace decomposition similar to PQ and apply a separate AQ quantizer in each subspace. Optimal rotation of OPQ may be applied to improve the allocations of dimensions in such case. This simple technique

is called *additive product quantization* (APQ). Additional benefit stems from the reduction in memory costs: storing M codebooks would normally require MKD memory, but if the decomposition is performed with S subspaces and $M' < M$ codebooks are used in each, the requirement becomes $S\left(M'K\frac{D}{S}\right) = M'KD$, plus D^2 (optionally) for the rotation matrix. APQ is quite commonly used instead of AQ in experiments with extremely large data (billions of vectors) or with high numbers of codebooks ($M \geq 16$).

3.3.3 Derivate methods

Several AQ derivate methods have been recently proposed [22][23]. While sharing the same representation and overall approach, these methods impose certain constraints on the codebooks, limiting the search space of encoding. This allows for significantly faster training and processing, making these solutions more practical. The cost of simplification is degradation in search performance, caused by the loss of generality. One such approach is described here to give an example of the tradeoff.

Composite quantization (CQ) [22] was first formulated in an attempt to simplify AQ distance calculations. As mentioned earlier, AQ distance estimate (14) differs from the one used in PQ (8) due to the presence of nonzero dot products between the codevectors. This scalar value could either be precomputed (and optionally quantized) or calculated during the search, requiring $M^2/2$ additional lookups. Composite quantization imposes the following constraint: for any set of M codevectors, each coming from a different codebook, their dot products have to sum up to a single constant value ϵ:

$$\forall c_1 \in C_1, c_2 \in C_2 \dots, c_M \in C_M\colon \sum_{i=1}^{M}\sum_{j=1,i\neq j}^{M}\langle c_i, c_j\rangle = \epsilon \tag{18}$$

In this case the corresponding component in (14) can be disregarded during the search, as it value is independent of both the query and the encoded data. Distance estimation then requires only M lookups from a query-specific table.

Besides the above simplification, CQ constraint carries a more important consequence. If it is perfectly satisfied, the reconstruction error of any database vector, which follows the same expression as (14), decomposes to sum of errors for individual codevectors. The encoding can then be performed in the same manner as PQ – by a series of independent nearest neighbor searches within each codebook. Still, even if $\epsilon = 0$, CQ remains more general than PQ, as no explicit subspace decomposition is performed and codebooks can be any orthogonal vector sets.

To satisfy the CQ constraint, the codebook adaptation procedure must be changed. The new error function is obtained by adding a quadratic penalty on constraint violation. Reformulated optimization problem of CQ training is given as follows:

$$\min_{C,a,\epsilon} \sum_n^N \left\| x_n - \sum_i^M c_{ni} \right\|_2^2 + \mu \sum_n^N \left(\sum_{i \neq j}^M \langle c_{ni}, c_{nj} \rangle - \epsilon \right)^2, \quad (19)$$

where μ is the penalty parameter chosen via cross-validation. Since this is a complex problem with no closed-form solution, trained composite quantizer will not exactly fulfill the CQ constraint in practice. Still, it is assumed that the solution is close to optimal and dropping the dot products in distance estimation (14) will not introduce significant errors.

Composite quantizer is trained by alternating between the following three steps:

1. *Updating assignments*, i.e. encoding the training set with current codebooks and fixed ϵ. Due to CQ constraint not being perfectly realized, independent codebook searches (akin to PQ) would produce suboptimal results, which can potentially propagate through training iterations. For this reason ICM (see Section 3.3.2) is adopted, as a compromise between simplicity and quality. However, instead of simple distances, (19) is used as a minimization criterion for choosing the codevectors. The complexity of this step is $O(LNMKD)$; authors suggest the number of ICM iterations $L = 3$.
2. *Updating* ϵ, given fixed codebooks and encoding. From (19) it trivially follows that an optimal value for ϵ is an average sum of dot products over the training set:

$$\epsilon = \frac{1}{N} \sum_n^N \sum_{i \neq j}^M \langle c_{ni}, c_{nj} \rangle \quad (20)$$

   The complexity of this update is $O(NM^2)$, assuming that the dot products have been precomputed and stored in a lookup table.
3. *Updating codebooks*, given encoding and ϵ. This is an unconstrained nonlinear optimization problem. CQ authors suggest the use of quasi-Newton solvers and specifically refer to L-BFGS, which has publicly available implementations [22].The complexity of this step is $O(NMDT_lT_c)$, where T_c is the number of L-BFGS iterations and T_l is a number of line searches within L-BFGS; CQ authors suggest $T_c = 10$ and $T_l = 5$.

At the beginning of each training iteration the dot products have to be computed and stored in a lookup table, to allow for fast objective function evaluation. The total asymptotic complexity of CQ training is thus $O(M^2K^2D + NMKD + NM^2)$.

For CQ there is no need to store dot product lookup tables or quantize the corresponding value, leading to some memory savings when compared to AQ. The two approaches are otherwise almost identical from the application viewpoint, after the training has been performed. However, CQ, unlike AQ, allows for fast vector encoding with ICM, which allows for smaller computational expenses of growing the database.

While the constraint on dot products reduces the representational ability of the quantizer, negatively affecting its performance metrics, it also allows for faster lookups and

much simpler encoding solutions to be viable. The example of CQ shows well how imposing constraints can ultimately be beneficial for the practical search applications.

Tree quantization (TQ) [23] is a direct successor to AQ, imposing very particular orthogonality constraints to make exact (optimal) encoding possible. In PQ a single dimension was represented by only one codebook, while in the opposite case of AQ it was represented by all the codebooks. TQ employs a compromise solution instead: every dimension is encoded exactly twice (see Figure 12).

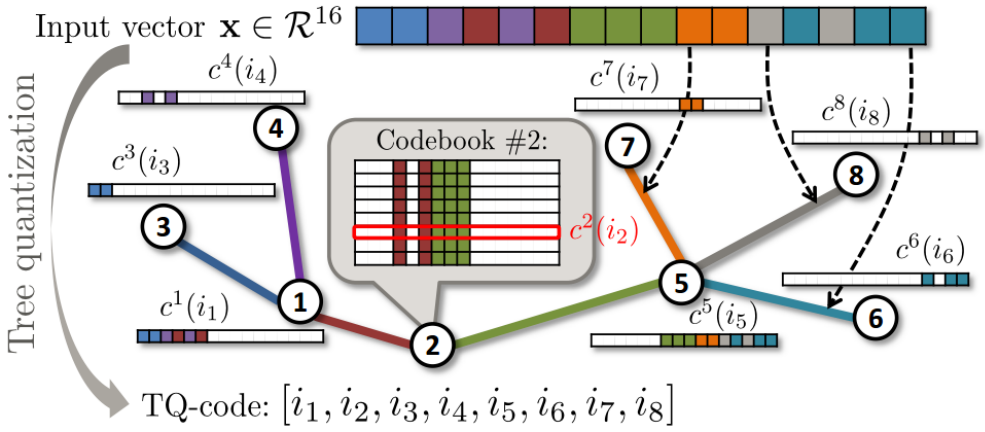

Figure 12. *Representation of TQ encoding with 8 codebooks* [23].

Underlying constraints are expressed with a tree structure, the so-called *coding tree*, where every node is a codebook and every edge corresponds to one or more data dimensions. Each codebook only contains dimensions that are mapped to its adjacent edges in the graph. It follows that any pair of codebooks which are not immediate neighbors in the tree are orthogonal. TQ thus performs subspace decomposition, but the one that is more general in nature.

While AQ encoding is equivalent to inference on fully connected MRF, TQ corresponds to its Chow-Liu approximation [23]. Exact solution can be calculated for the latter in polynomial time; in practice this translates to $O(MK^2)$ encoding complexity. Another benefit is that during the distance estimation, the dot product component of (14) can be calculated in linear time, only being nonzero for the codebooks adjacent in the tree.

Similarly to CQ, TQ training scheme has to be extended to accommodate for constraints. Between the encoding, which is done with exact inference on the graph, and the codebook adaptation, which does not differ from AQ, the coding tree itself needs to be constructed. A complex integer linear programming (ILP) problem is formulated, with large set of constraints ensuring the proper tree structure (e.g. absence of loops). The authors of the method have used a commercial general purpose solver for the latter, meaning that no complexity analysis is possible for training [23].

4. FAST ENCODING FOR ADDITIVE QUANTIZATION

It has been empirically noticed that the reduction in quantization error typically leads in search performance improvements [18]. In addition, the quantization error on the training set is typically representative of the whole database, more so when the size of the training data increases. Taking these assumptions can significantly simplify the development and analysis of new techniques, as quantization error is typically well-defined and easy to examine. The same assumptions are thus taken for derivation of the proposed methods. Before the detailed description is given, it is beneficial to examine the general behavior of the quantization error for additive representations. As no global optimization techniques have been found suitable for AQ yet, only incrementally learned solutions are considered.

4.1 Quantization error of incremental solutions

The complexity of AQ encoding stems from the presence of dot product sum component $\sum_{i=1}^{M}\sum_{j=1,i\neq j}^{M}\langle c_i, c_j\rangle$ in (14). It invalidates greedy approaches by introducing interaction between codevectors. The dot product component can also have a negative sign, unlike squared norms. Since encoding is performed iteratively, by adding one codebook index at a time, it is possible that a seemingly suboptimal choice at one time instance can lead to a larger error reduction later.

One solution is to constrain the dot products, simplifying the optimization procedure and thus improving the greedy solutions. Among the methods considered so far, (O)PQ has strictly orthogonal codebooks, CQ imposes the constant value on the dot products, and TQ retains only some nonzero values. All of these approaches, however, sacrifice the generality of the representation, leading to larger quantization error compared to AQ. Considering this, it might be reasonable to avoid placing explicit constraints and instead seek for a different optimization procedure.

Consider a database vector x and its additive reconstruction C_{prev}, with the quantization error $E_0 = \left\|x - C_{prev}\right\|^2$. Assume that another codevector C_{next} is added to the representation, resulting in a new error value $E_1 = \left\|x - (C_{prev} + C_{next})\right\|^2$. This expression can be expanded as follows:

$$\left\|x - (C_{prev} + C_{next})\right\|^2 = \|x\|^2 + \left\|C_{prev} + C_{next}\right\|^2 - 2\langle x, C_{prev}\rangle - 2\langle x, C_{next}\rangle \quad (21)$$

The second squared norm in the right-hand part can be expanded further:

$$\left\|C_{prev} + C_{next}\right\|^2 = \left\|C_{prev}\right\|^2 + \|C_{next}\|^2 + 2\langle C_{prev}, C_{next}\rangle \quad (22)$$

Meanwhile, the sum of quantization errors, if C_{prev} and C_{next} are used individually, would be

$$\left\|x - C_{prev}\right\|^2 + \|x - C_{next}\|^2 = 2\|x\|^2 + \left\|C_{prev}\right\|^2 + \|C_{next}\|^2 - \\ -2\langle x, C_{prev}\rangle - 2\langle x, C_{next}\rangle \quad (23)$$

Combining (21), (22) and (23) leads to the following expression:

$$\left\|x - \left(C_{prev} + C_{next}\right)\right\|^2 = \left\|x - C_{prev}\right\|^2 + \|x - C_{next}\|^2 - \\ -\|x\|^2 + 2\langle C_{prev}, C_{next}\rangle \quad (24)$$

Since $\left\|x - C_{prev}\right\|^2$ is the quantization error of the original representation, the *error difference* ΔE after the refinement is

$$\Delta E = E_1 - E_0 = \|x - C_{next}\|^2 - \|x\|^2 + 2\langle C_{prev}, C_{next}\rangle \quad (25)$$

Using the geometric definition of the dot product and expressing the squared norms through dot products, the alternative expression for ΔE is obtained:

$$\Delta E = \|C_{next}\|^2 - 2\|x\| \cdot \|C_{next}\| \cdot \cos\alpha + 2\left\|C_{prev}\right\| \cdot \|C_{next}\| \cdot \cos\beta \quad (26)$$

Here α is an angle between vectors x and C_{next}, while β is an angle between C_{prev} and C_{next} (see Figure 13 for a simple 2-dimensional visualization).

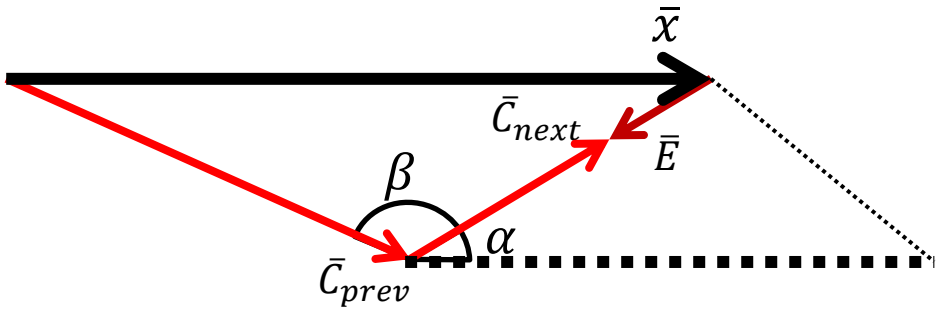

Figure 13. *Refining the additive vector representation.*

With a greedy encoding procedure it is expected that the quantization error will always decrease for a good choice of C_{next}. To achieve maximal possible reduction, the following criterion needs to be optimized:

$$\min_{\|C_{next}\|,\alpha,\beta} \|C_{next}\| - 2\|x\|\cos\alpha + 2\left\|C_{prev}\right\|\cos\beta. \quad (27)$$

Expression (27) was obtained from (26) by discarding the multiplier $\|C_{next}\|$, which does not affect the semantics, since it is strictly positive and ΔE is expected to be negative for a good choice of C_{next}.

Problem (27) shows the factors taken into account, when a greedy selection of a new codevector is performed. Every term can be interpreted to have its own meaning:

1. $\|C_{next}\|$ is the norm of the newly chosen codevector. Being strictly positive (as adding null vectors is meaningless), it ensures the *minimality of the representation* – codevectors of smaller length are preferred, other criteria notwithstanding.
2. $\|x\|\cos\alpha$ is the scalar projection of the new codevector on the database vector. This is the *fitting term*, which maximizes the contribution of the new codevector to the approximation. Note that the new codevector is fit to the target vector independently from previous choices. The sign of the term is not restricted, so it is possible to increase the quantization error with a poor choice.
3. $\left\|C_{prev}\right\|\cos\beta$ is the scalar projection of the new codevector on the current reconstruction. This is the *correction term*, as it can also take any sign and can contribute to the error reduction by interacting with the fitting term.

The geometric interpretation of the simplest greedy solution (choosing a single best codevector from a set) is apparent from (27): the current reconstruction and the target are both projected on each candidate codevector, with resulting scalars composing the "fitness" of this particular solution.

The orthogonality of the codebooks would result in $\cos\beta = 0$, leading to the absence of the correction term. This limits the possible options for C_{next}, reducing the expressiveness of the representation. For example, with non-orthogonal codebooks it is possible to include a codevector of the opposite direction to x ($\cos\alpha < 0$) and still achieve error reduction, if $\cos\beta > 0$. The advantage of AQ over PQ in terms of iterative encoding thus stems from the fact that the later codevector choices are capable of correcting the previous approximations.

One possible direction of future work is to consider quantizing the norm and the direction of the database vectors separately. The norm, being a scalar value, is efficient to quantize; the expressions (21)–(27) can then be simplified and analyzed from the purely geometrical viewpoint, with angles playing a major role. This work instead focuses on the merging of several existing encodings.

4.2 Pyramid encoding

Both ICM and Beam Search are iterative encoding procedures, performing successive local searches to locate suitable codevectors. The vast discrepancy in their effectiveness and computational complexity is caused by two major differences – size of local search space and storing several candidate solutions to alleviate greediness. ICM only considers one codebook at a time, resulting in K codevector choices at each step. The ordering of codebooks is arbitrarily fixed and does not change within the training procedure. No candidate solutions are kept: a single encoding is optimized instead. Beam Search scans the entirety of codevectors, excluding, of course, codebooks which are already used. The number of codevector choices thus starts with MK and gradually decreases. In addition, a certain number of candidate encodings is stored for the 2-step optimization.

With ICM and Beam Search representing the opposite extremes, the compromising solution would keep the middle ground in both aspects. Keeping a pool of solutions between iterations partially alleviates the local nature of the search, without much increase in complexity (e.g. Beam Search is linear in search depth H); it is thus very useful and is retained for the proposed methods. Properly constraining the search space, however, warrants a closer examination.

Chapter 4.1 contains the derivation (21)–(24) of the quantization error value, when an existing reconstruction C_{prev} is extended with a new codevector C_{next}. However, the nature and internal structure of C_{prev} and C_{next} are completely arbitrary. The same expressions can be applied to find the quantization error when two different encodings are merged together.

Assume that a database vector x is encoded with two separate AQ quantizers (their parameters are irrelevant at this point), resulting in 2 approximations: $\tilde{x}_1$ and $\tilde{x}_2$, with quantization errors E_1 and E_2 respectively. Then from (24) it follows that the error of their combination is:

$$E = \|x - (\tilde{x}_1 + \tilde{x}_2)\|_2^2 = E_1 + E_2 - \|x\|_2^2 + 2\langle \tilde{x}_1, \tilde{x}_2 \rangle \tag{28}$$

Again, if the quantization error is only required for the purpose of ranking different solutions, the term $\|x\|_2^2$ may be omitted without consequence. Only dot product between $\tilde{x}_1$ and $\tilde{x}_2$ is therefore required to obtain a new error value. Assuming that the dot product lookup table is available and both quantizers in question have M codebooks, combining them would cost only M^2 table lookups.

Pyramid encoding is a proposed encoding method for AQ, which utilizes the above scheme to hierarchically merge solutions until the required number of codebooks is reached. The merging is implemented in a hierarchical structure, best represented with a tree (see Figure 14; dashed lines indicate the search directions). The name “pyramid

encoding" is adopted to avoid confusion with tree quantization (TQ), which is a fundamentally different approach [23].

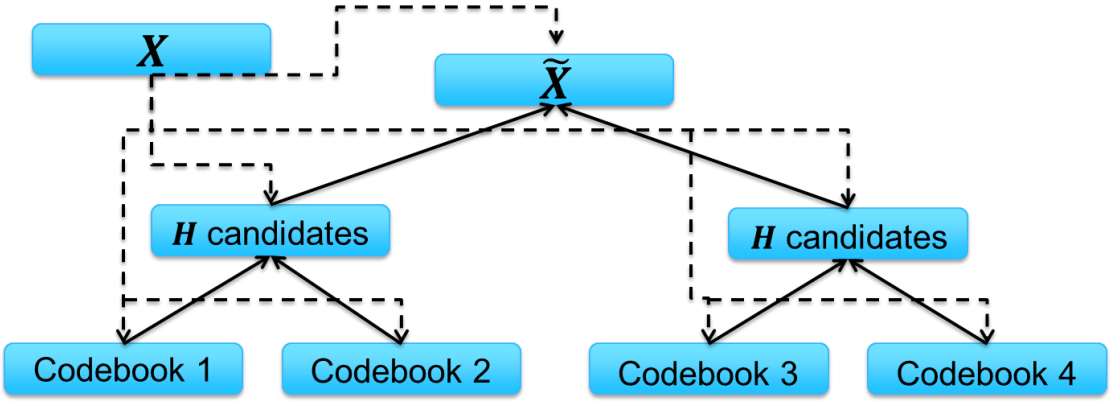

Figure 14. *The structure of pyramid encoding for 4 codebooks.*

Before the encoding process can begin, the dot products between all the pairs of codebook vectors are precomputed and stored in a lookup table, which takes $O(M^2K^2D)$ operations. Since it does not depend on the vector being encoded, this table can be reused as long as the codebooks are not changed. The precomputation cost, being independent from N, becomes negligible when processing large datasets.

Individual codebooks form the bottom level of the pyramid. At this point distance calculations are unavoidable. Nearest neighbors for a database vector x are located in each codebook separately, resulting in M searches requiring KD operations each, for a total of MKD multiplications and additions. However, these are the only required calculations that depend on the data dimensionality, as error update (28) uses only dot product lookups.

As the encoding proceeds, pairs of solutions are merged on each level of the pyramid. Some ordering of the codebooks is required to make this possible. One option is to perform an additional search to find the pairs of solutions with smallest error values. However, this dramatically increases the computational costs involved (up to M^4 in some scenarios). An arbitrary ordering is used instead, similarly to ICM. It is plausible to assume that during the training the codebooks will be adapted to fit whatever ordering is chosen.

The number of possible combinations of solutions grows dramatically as the pyramid is ascended. The bottom level has K options in each tree node, but the level above has K^2, as pairs of codebooks are connected. This is followed by K^4 and so on, until the top level is reached, where all the codebooks are used together for the K^M possible options, which is the same as the original unconstrained NP-hard optimization.

To alleviate the explosive growth of solutions, the number of candidates is artificially limited on each pyramid level, via a cutoff procedure. In each node, including the bottom level, only H best solutions are kept, where H is a user-provided parameter called *search depth* (as its meaning and purpose closely relate to those of the Beam Search parameter). In this case there are H^2 candidates to consider in each merging node, from which, again, the best H are retained for the next merge. This limitation naturally makes the final solution approximate, but allows for manageable computational costs.

As mentioned earlier, the errors throughout the pyramid can be calculated from (28) using table lookups. Each level has 2 times less nodes than the previous one, as the solutions are merged pairwise. However, during ascent the number of required lookups also increases. Consider two solutions using one codebook each: $\{c_1\}$ and $\{c_2\}$. To find the quantization error of their combination, one needs to calculate the dot product $\langle c_1, c_2 \rangle$, requiring a single table lookup. However, when the result $\{c_1, c_2\}$ is merged with another solution $\{c_3, c_4\}$ on the higher hierarchical level, the dot product becomes $\langle \{c_1, c_2\}, \{c_3, c_4\} \rangle = \langle c_1 + c_2, c_3 + c_4 \rangle = \langle c_1, c_3 \rangle + \langle c_1, c_4 \rangle + \langle c_2, c_3 \rangle + \langle c_2, c_4 \rangle$: equivalent to 4 table lookups.

Treating the number of lookups per level as a geometric series and noting that the pyramid height is equal to $\log_2 M$, it is possible to find the complexity of all the merging operations together. The result is $O(M^2H^2)$, leading to the total complexity of encoding a single database vector with the proposed method: $O(M^2H^2 + MKD)$. The pyramid encoding is thus significantly faster then Beam Search, considering that the same value of H (e.g. $H = 64$) is perfectly applicable. Disregarding the lookup table construction cost (which does not change between methods), the pyramid encoding roughly attains a speedup of factor MK/H. For instance, with $M = 8$ codebooks of $K = 256$ codevectors each, setting the search depth to $H = 64$ results in pyramid encoding being approximately 32 times faster.

Despite the computational gains, in terms of quantization error and search performance it is reasonable to expect the inferiority of the pyramid encoding as compared to the Beam Search. As mentioned earlier, the original AQ encoding searches a larger space on each step and is not constrained to any specific ordering of the codebooks.

4.3 Residual pyramid encoding

The pyramid encoding scheme described above fits each individual solution to a single target – the database vector x. Assume that 2 independent additive approximations of x have been learned – $\tilde{x}_1$ and $\tilde{x}_2$. If both solutions are reasonably accurate, it follows from definition that $\tilde{x}_1 \approx x$ and $\tilde{x}_2 \approx x$. When such a pair is merged in the pyramid encoding, the result is $\tilde{x}_1 + \tilde{x}_2 \approx x + x \approx 2x \neq x$. Combining the two solutions, both close to optimality when taken individually, can thus result in an error increase. Furthermore, as

this holds true for all the levels of the pyramid, the discrepancy accumulates, potentially hindering the search.

One possible approach to tackling this problem is to disregard it. Since the codebook adaptation step is consistently applied on each training iteration, it can be reasonable to expect that the codebooks will gradually adjust to the pyramid structure. However, assuming that the adaptation step is not changed, the linear systems solved to obtain the codevectors contain no explicit information about the pyramid. There is no way to prove that the adaptation will happen. There are also no guarantees that, if the learning does indeed happen, it will not be inexplicably slow and/or reliant on random factors (e.g. initialization). However, it is possible to reformulate the pyramid encoding to take the factors described above into account.

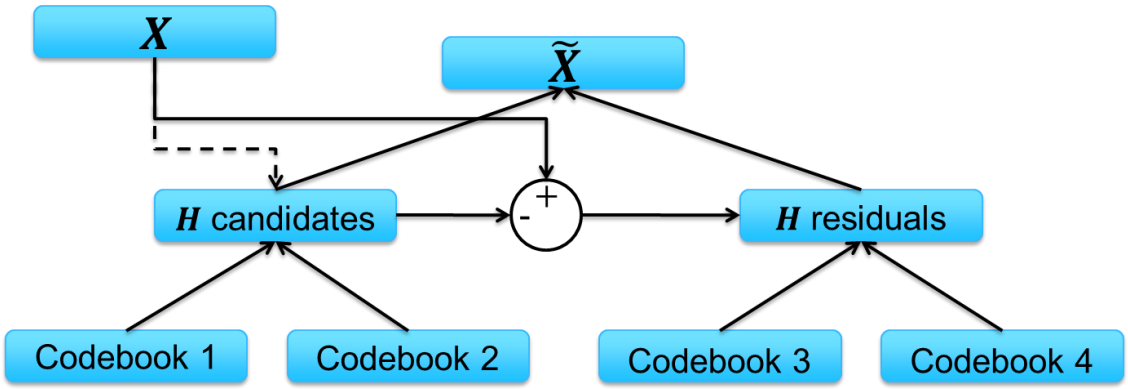

Figure 15. *Residual pyramid encoding for one subspace.*

Residual Pyramid Encoding (RPE) is a variant of the pyramid encoding, where the right branch of the final solution is fitted to the residual of the left branch and not to the target vector (see Figure 15). While this formulation is trivial in case of 4 codebooks and 3 hierarchical levels, unlike the pyramid encoding, there is no straightforward generalization to the cases where $M > 4$. It is thus necessary to perform subspace decomposition (similarly to PQ) and apply residual pyramid encoding with4 codebooks individually on each subset of dimensions. For example, if $M = 16$, the vector would be split into 4 subvectors, each processed with a separate quantizer. This additional constraint is likely to have a negative impact on the search performance of the resulting representation, as discussed with respect to PQ (see Chapters 3.2–3.3).

To compensate for the limitations on the number of codebooks, no cutoff is performed on the bottom level (otherwise the representation accuracy drops significantly). The residual pyramid encoding thus starts by evaluating K^2 candidates from a pair of codebooks. From these the top H with smallest quantization errors are retained. H residual vectors are then calculated by subtracting the representations of the previous step from the target vector. Finally, a second pair of codebooks (again, having K^2 possible combi-

nations) is analyzed to produce the best fit for the residuals. The one with the lowest overall error is chosen as a final solution for the target vector x.

The complexity of a single realization of the residual pyramid encoding is $O(K^2H)$. However, due to a need for subspace decomposition, $M/4$ residual pyramids need to be calculated to encode a single vector. Since a lookup table is also necessary, the total complexity of the residual pyramid encoding becomes $O(MK^2H)$. The costs of this method are noticeably higher than the ones for the simple pyramid encoding, especially if larger values of search depth parameter are used. Experimental assessment is required to conclude if the quality of encoding warrants the complexity increase.

4.4 Codebook diversity problem and initialization

The combination of a pair of approximations does not necessarily improve upon their individual errors. One example of such behavior is the overestimation of the target vector norm, described in the previous section as a motivation for the residual pyramid encoding. However, it is just one among the possible manifestations of a more general issue – *codebook diversity problem*.

The codebook diversity problem can be described as follows. When two additive vector representations are combined, any information about the target vector that is covered by both of them is redundant (and might even lead to overestimation). Only the unique aspects of each individual solution contribute to the possible error reduction. The quantizer should therefore maintain the diversity between codebooks; otherwise the pyramid encoding formulation is suboptimal. Codebook redundancy greatly reduces the representational ability of the quantizer. The result is an overall efficiency loss, as increasing the number of codevectors raises computational costs, but does not significantly affect the quantization error.

Again, it is possible to disregard the problem, relying on the codebook adaptation to provide a natural solution over time. However, as the optimization employed is very local in nature, the codebooks do not exhibit dramatic changes from their starting state, which is given by pre-encoding initialization. As a consequence, the training based on pyramid encoding is very sensitive to the initial choice of codebooks.

The traditional approach to initialize an AQ learning process is to randomly generate the codes for the training set, sampling them from the uniform distribution (M numbers per vector, each in range $[1, K]$). The training itself then begins from the codebook adaptation. However, the uniformity of codes results in mostly uniform codebooks. Beam Search, as stated by its authors [21], does not exhibit much dependence on a starting point, but pyramid encoding is hindered by lack of diversity. A different initialization approach would greatly benefit the performance of suggested methods.

Product quantization is an approach where codebook diversity is explicit and extreme. Since codebooks encode non-intersecting parts of the target vector, any redundancy between them is impossible. PQ can therefore be used as an initial estimator, before the proper AQ training is started. Since the computational costs of PQ are much lower, and the procedure needs only to be run once using a small number of iterations, the complexity increase can be disregarded. After the initial PQ processing is completed, AQ codebook adaptation is performed on the codes. The normal iterative training procedure follows.

Since there are no constraints imposed on codevectors in AQ, the orthogonality of codebooks is not perfectly maintained and is typically lost after the first iteration of training. This is beneficial, as orthogonality was earlier shown to restrain the representations. However, low interdependence of solutions means that merging is almost always effective in terms of error reduction. Thus it can be argued that AQ with pyramid encoding represents a "compromise" between PQ and original AQ, resembling methods such as CQ and TQ. However, these methods explicitly state and learn their constraints, while AQ with pyramid encoding and PQ initialization implicitly relaxes the codebook orthogonality to reduce the error.

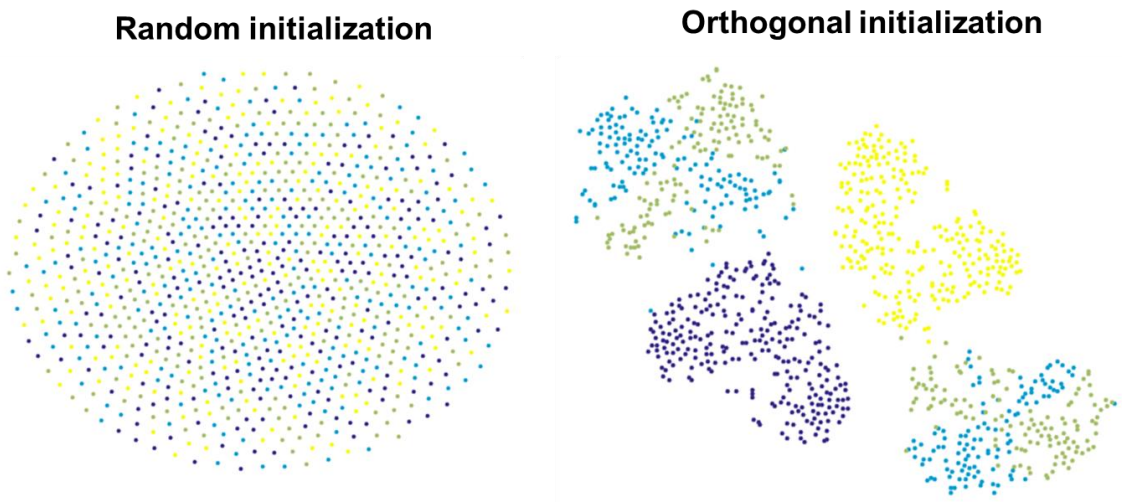

Figure 16. *t-SNE embedding of pyramid encoding codebooks.*

To illustrate the above statements, AQ quantizers ($M = 4,\ K = 256$) were trained with pyramid encoding on the set of 100 000 SIFT image descriptors [42]. Both random initialization and PQ were used to obtain starting codes. The resulting 128-dimensional codebooks were then mapped to 2-D space with *t-Distributed Stochastic Neighbor Embedding* (t-SNE). This well-known dimensionality reduction technique is capable of preserving both local and global structure, making it suitable for data visualization [43]. The result is shown on Figure 16, where each codebook is color-coded. It is apparent that the PQ initialization allows codebooks to specialize, while learning from random codes leads to the lack of diversity.

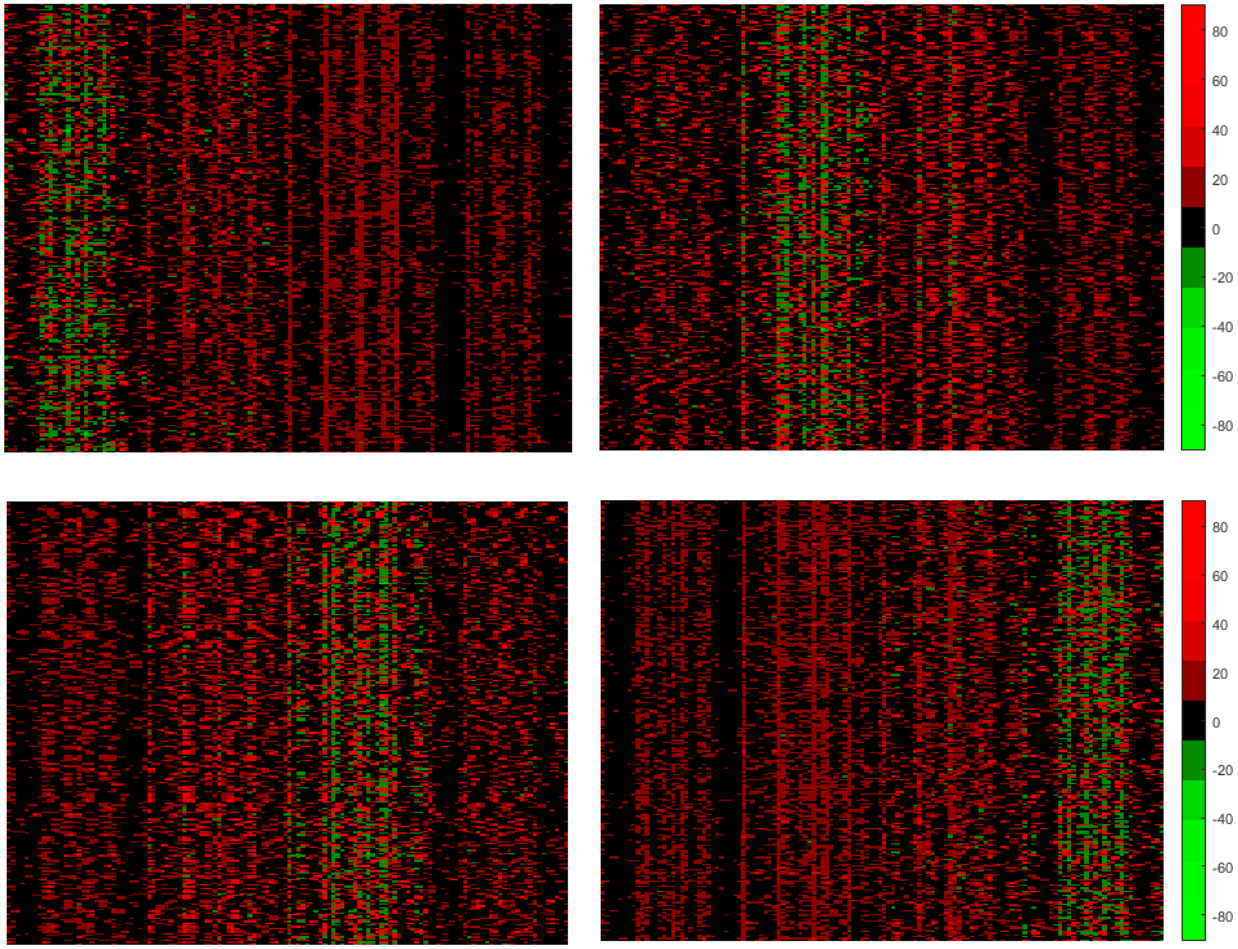

Figure 17. *Heatmaps of the codebooks learned with PQ-initialized pyramid encoding.*

Figure 17 shows the heatmaps of the PQ-initialized codebooks after the training. While strict orthogonality no longer holds, the remnants of the original subspace decomposition are immediately obvious. These unique structures result in less redundancy than otherwise achievable.

Naturally, the same considerations apply to residual pyramid encoding. The random initialization is even less suitable here, as it does not take into account the fact that some codebooks encode residuals of others. This, in turn, might lead to the failure of residual pyramid encoding to exhibit its beneficial properties. A number of other approaches are suggested below.

Residual initialization for RPE is performed as follows. A set of codes for the first two codebooks, which encode the target vector directly, are initialized randomly. Then the linear equations (16) are solved for these codes only, resulting in two complete codebooks. The reconstructions for the training data are then calculated and subtracted from it to obtain a set of residuals. The second pair of codebooks is then calculated from these residuals and uniform random codes.

Orthogonal initialization (or *PQ initialization*) applies the product quantization on the data and uses the resulting codes for the first adaptation step. This is no different from the PQ initialization of pyramid encoding and can therefore serve as a point of comparison.

Hybrid initialization combines the two approaches suggested above. First PQ is applied on the data, but only 2 codebooks (subspaces) are used. The reconstructions are then subtracted from the data to obtain a set of residuals, which are encoded in the two remaining codebooks (from random codes).

While it is possible to employ other quantization methods for initialization purposes, it is undesirable due to their higher costs and no strong guarantees of codebook diversity. One exception from this is optimized product quantization (OPQ), that features a relatively minor overhead when compared to PQ, as well as much smaller reconstruction error. Adaptive allocation of dimensions to subspaces may also prove beneficial. However, experiments failed to show any significant advantage of OPQ initialization, with most quantizers ultimately converging to the PQ-based solution. For these reasons no other initialization methods are considered in the experimental setting.

5. EXPERIMENTAL RESULTS

5.1 Experimental procedure and datasets

Vector quantization-based ANN methods typically share the same experimental workflow. The benchmark dataset is split in two non-intersecting parts; one of these, denoted as a *hold-out set*, is used in the iterative training procedure to obtain a good set of codebooks. A hold-out set should be not only representative, but also small enough to facilitate faster training. The second part of the original data (the *base set*) is then encoded with trained codebooks. At this point the *quantization error* value can be used to estimate the representation accuracy of the quantizer; there is empirical evidence that lower quantization error corresponds to better search performance [18].

To calculate the recall, a *query set* is used, for which the *ground truth* is available. Ground truth refers to known indices of true nearest neighbors to each query vector amongst the vectors of the base set. Since it's common to use only a small number of true nearest neighbors to measure performance, usually the ground truth contains only about a hundred indices for each query vector. In this work the recall values are calculated for the first true neighbor only, making recall@T equivalent to *precision* for a given window size T.

Two datasets were used for experiments – *SIFT1M* and *GIST1M*. Both having been introduced in [18], they became standard benchmarks in the field. Information about both datasets is provided in Table 1.

Table 1. *Details of datasets used in experiments*

	SIFT1M	**GIST1M**
Content	SIFT image descriptors [44]	GIST image descriptors [35]
Dimensionality	128	960
Base set size	1 000 000	1 000 000
Hold-out set size	100 000	500 000
Query set size	10 000	1 000
Ground truth length	100	100

Since both datasets are comprised of image descriptors, calculated from images freely available online, they can be considered a reasonable approximation for an image retrieval application. All the data is openly accessible from [42].

The experiments were conducted with product quantization (PQ), optimized product quantization (OPQ) and additive quantization (AQ), to provide an indicative comparison for the proposed improvements. For AQ using the original Beam Search encoding the label AQ-B is used throughout the following sections. AQ with proposed pyramid encoding is denoted as AQ-P when used random initialization and as AQ-P(pq) when using orthogonal initialization. For the residual pyramid encoding the abbreviation AQ-RP is used, followed by (pq) for orthogonal initialization, (rs) for residual initialization and (h) for hybrid initialization. PQ, OPQ and AQ with non-standard encoding were implemented by the authors; the first two were verified against results described by their original creators. For AQ with Beam Search encoding a well-optimized publicly available implementation was used [45].

Table 2 lists the parameters used for the quantizers. Same values are used for all the experiments, unless stated otherwise.

Table 2. *Parameter values for different quantizers.*

Parameter	Value
Number of codebooks M	4 and 8
Number of codevectors per codebook K	256
Representation codelength	32 bits for 4 codebooks, 64 bits for 8 codebooks
Search depth H	AQ-B – 16 for training, 64 for base encoding AQ-P – 64 AQ-RP – 16
Number of training iterations	30 for SIFT1M 10 for GIST1M
Number of PQ iterations for initialization of AQ-P(pq), AQ-RP(pq), AQ-RP(h)	15

The search depth H for pyramid encoding and residual pyramid encoding was chosen empirically, based on small-scale experiments with 10%-subsets of data (both SIFT1M and GIST1M were considered). It was found that $H = 64$ is a suitable value for the residual encoding, as setting it higher did not have a significant performance benefit, while decreasing it lead to noticeable increase in quantization error. AQ-RP was found to be quite robust in terms of H, so smaller value was chosen to partially mitigate the increased encoding costs.

To evaluate search performance, recall@1, recall@10 and recall@100 are computed on each dataset; on GIST1M recall@1000 is also shown. For a more complete comparison also the performance of several other recent methods is provided. No experiments are performed with these, their reported results from respective papers are used instead. In cases where a particular value was not reported by the original authors, it is omitted. The list of these methods is as follows:

- *Optimized Cartesian K-means* (OCKM) [46] is a modification of CKM (OPQ), where every codebook is split into two additive subcodebooks to allow for more representation flexibility.
- *Composite Quantization* (CQ) [22] is a variant of AQ, described in Chapter 3.3.3.
- *Locally Optimized Product Quantization* (LOPQ) [39] is a variant of OPQ, briefly described in Chapter 3.2.3.
- *Stacked Quantization* (SQ) [47] is a hierarchical approach, where the first codebook is adapted to fit the original database vector, the second codebook is learned on a set of residuals and so on.
- *Optimized Tree Quantization* (OTQ) [23] is a modification of TQ, which was briefly described in Chapter 3.3.3. OTQ additionally learns and applies rotation on the data, following exactly the steps of OPQ.

5.2 Computational complexity

Any comparison between the ANN methods needs to take into account not only the search performance, as measured by recall, but also the computational complexity. For all the compared methods the codebook adaptation costs are negligible compared with encoding costs. For this reason only the complexity of encoding a single vector is considered. The estimates of the original algorithm authors were used where possible, otherwise the derivation was made as part of thesis work. The results are provided in Table 3.

Table 3. *Complexity of encoding a single database vector with different quantizers*

Quantization method	Complexity
PQ	$O(KD)$
OPQ	$O(KD + D^2)$
AQ	$O(M^3KH + MKD)$
OCKM	$O(10MKD)$
CQ	$O(3MKD)$
LOPQ	$O(KD + D^2)$
SQ	$O(M^2KD)$
OTQ	$O(MK^2 + KD + D^2)$
AQ-P	$O(M^2H^2 + MKD)$
AQ-RP	$O(MK^2H + MKD)$

Numeric values for OCKM and CQ entries are obtained by substituting the extra parameters with their recommended values [46][22]. LOPQ has an additional sizeable overhead when compared to OPQ, but it is constant in data size [39].

PQ, OPQ and LOPQ have a significantly smaller encoding complexity, when compared to other methods. Subspace decomposition is performed in a way that makes the costs independent from the number of codebooks M. This allows the codelength to scale up

for improved representation without any cost increase during training or encoding (although distance computations will naturally become slower, as there is more table lookups involved). AQ encoding (Beam Search), as mentioned earlier, features prohibitive complexity, especially for larger codelength, as it is cubic in the number of codebooks. Other methods (OCKM, CQ, SQ and OTQ) provide significant improvements, with encoding costs being quadratic or even linear in terms of M and K.

As can be seen from the table, pyramid encoding complexity compares favorably to the others. Although many other methods scale better (linearly) with an increase in M, decoupling the data dimensionality D from the other parameters (due to lookup tables) reduces the negative impact of large vector lengths on computation. Methods such as OCKM, SQ and CQ may prove inferior to AQ-P in such scenario. In addition, under the reasonable assumption that $H < K$, pyramid encoding can be expected to achieve faster encoding than OTQ. Computational superiority in comparison with AQ was already discussed in the previous chapters.

Residual Pyramid encoding (AQ-RP) does not compare as favorably to other methods. Because of subspace decomposition the constant coefficient of the number of operations can be expected to be small, and the algorithm scales linearly with the number of codebooks, but otherwise its computational benefits can only manifest on high-dimensional data and only in comparison with methods without table lookups (e.g. SQ, CQ).

5.3 SIFT1M results and comparison

Figure 18 shows the training curves of PQ, OPQ and AQ with different encoding types on the SIFT1M dataset for 2 codelengths – 32 bits, corresponding to 4 codebooks, and 64 bits, corresponding to 8 codebooks. Figure 19 shows the quantization error achieved on encoding the base set for all the proposed encoding types and initializations.

Pyramid encoding with random initialization (AQ-P) provides a more accurate representation than PQ and OPQ, but is outperformed by AQ-B. With orthogonal initialization, however, best result so far is attained, with especially significant error reduction for small codes (32 bit). AQ-P(pq) also shares with PQ the fastest convergence on both codelengths, rivaled only by AQ-B in 64-bit case.

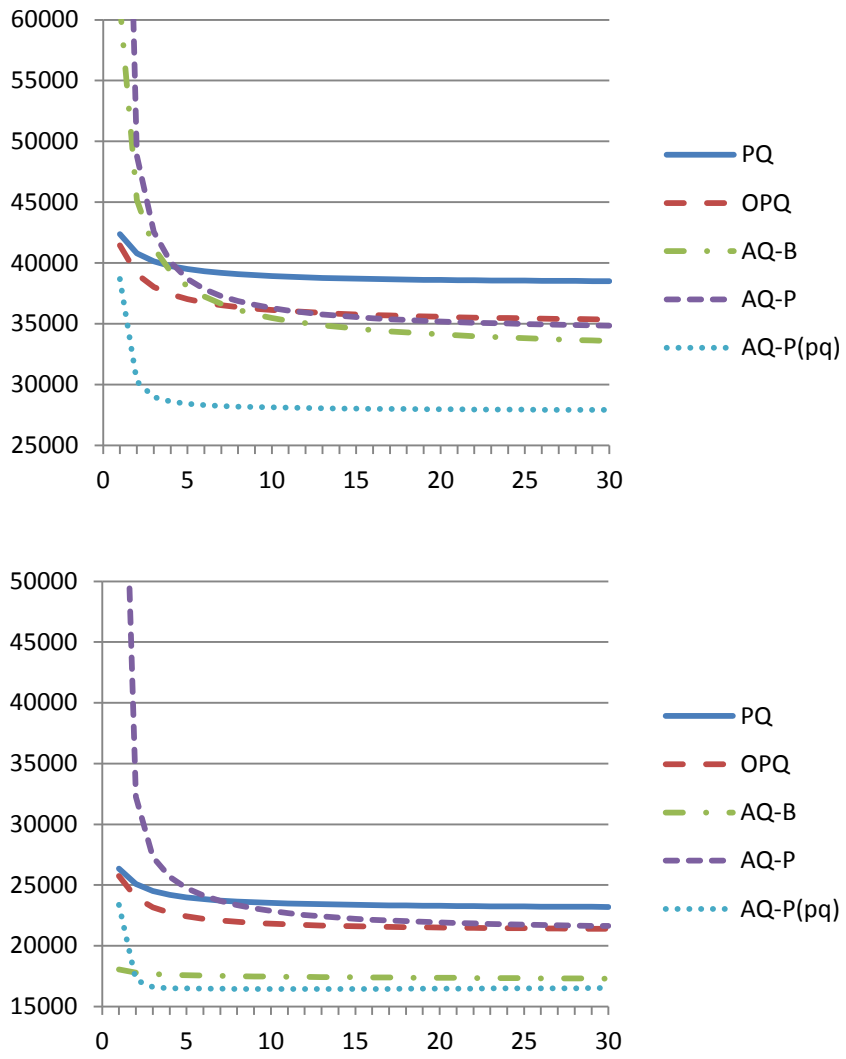

Figure 18. *Training error on SIFT1M for 32 bit codes (top) and 64 bits (bottom)*

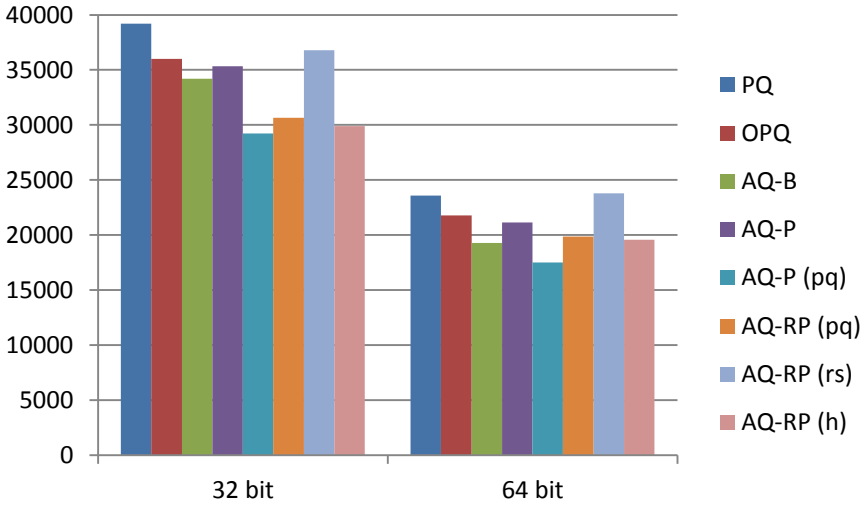

Figure 19. *Quantization error on SIFT1M base set.*

Despite this, it is important to reiterate on the fact that the quantization error is not always representative of search performance. For this reason the recall values at 3 cutoff points (1, 10 and 100 top vectors) were calculated for the abovementioned methods. These results are listed in Table 4. Additional results from other competing methods, as reported by their respective papers, are also provided for reference. Unreported values are marked with a dash. For every cutoff point the highest and the second highest recall values are highlighted for clarity.

Table 4. *Recall@T values on SIFT1M*

64 bits (M = 8)	**R @1**	**R @10**	**R @100**	**32 bits (M = 4)**	**R @1**	**R @10**	**R @100**
PQ	0,2243	0,599	0,9243	**PQ**	0,0518	0,2297	0,5945
OPQ	0,2433	0,6384	0,9402	**OPQ**	0,06756	0,2725	0,6576
AQ-B	***0,3114***	**0,7733**	**0,983**	**AQ-B**	0,1066	**0,415**	**0,8255**
OCKM	0,2735	0,6804	0,9448	OCKM	–	0,348	0,742
CQ	0,288	0,7159	0,9666	CQ	–	–	–
LOPQ	0,2972	0,7031	0,9577	LOPQ	**0,1342**	0,3858	0,738
SQ	0,2756	0,6942	0,9621	SQ	0,094	0,3445	0,7342
OTQ	**0,317**	***0,748***	***0,972***	OTQ	0,093	0,368	0,793
AQ-P	0,2051	0,5808	0,9115	***AQ-P***	0,0560	0,2438	0,6248
AQ-P (pq)	0,2846	0,7125	0,9661	***AQ-P (pq)***	***0,1177***	***0,3983***	***0,7941***
AQ-RP (pq)	0,2631	0,6815	0,955	***AQ-RP (pq)***	0,0963	0,3574	0,7411
AQ-RP (rs)	0,2151	0,5882	0,9165	***AQ-RP (rs)***	0,0475	0,2229	0,5982
AQ-RP (h)	0,2787	0,6956	0,9614	***AQ-RP (h)***	0,1162	0,397	0,7866

AQ-P with random initialization does not demonstrate a very good search performance, barely surpassing PQ. Pyramid encoding with orthogonal initialization, however, achieves second best results (after AQ and LOPQ) for 32-bit codes and remains very competitive with other approaches with 64-bit codes. As discussed earlier, it also has an advantage in computational costs, making it a very viable approach, especially with small number of codebooks.

Residual pyramid encoding (AQ-RP) is generally inferior to other methods, especially when its complexity is taken into account. It does achieve results comparable with AQ-P when hybrid initialization is used, but otherwise there is no apparent benefit from its use.

5.4 GIST1M results and comparison

GIST1M data has a significantly higher dimensionality when compared to SIFT1M (960 against 128), as well as a larger hold-out set (500 000 against 100 000). This leads

to large amounts of computation involved in any experiments, especially for quantizer training. Consequently, experiments with GIST1M were limited to the most promising approaches, as inferred from the preliminary estimates on smaller data subsets. Unfortunately, this also applies to the authors of competing methods, meaning that fewer performance results are reported, especially for algorithms with high encoding complexity.

Figure 20 shows the quantization error of PQ, OPQ and AQ with different encoding types on the GIST1M dataset for 2 codelengths – 32 bits, corresponding to 4 codebooks, and 64 bits, corresponding to 8 codebooks.

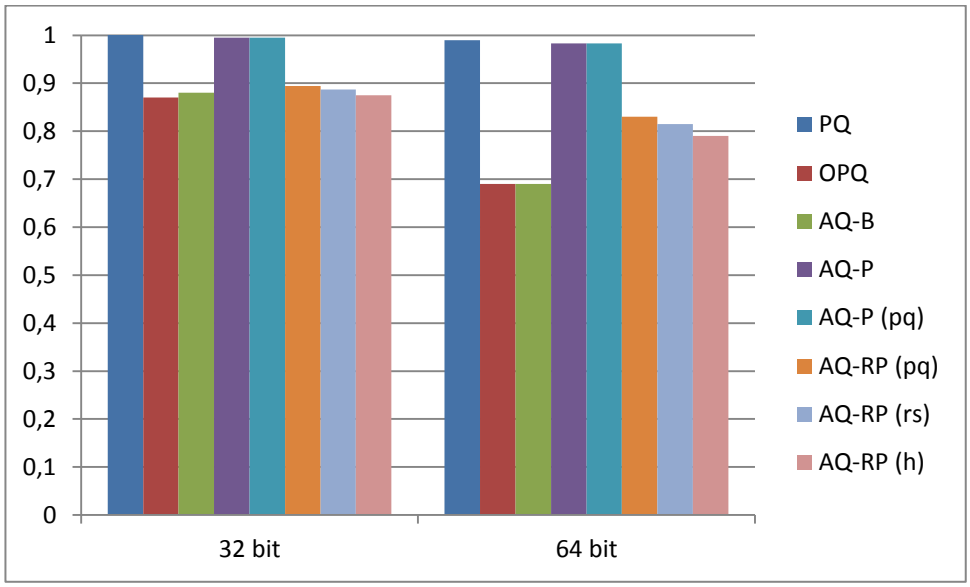

Figure 20. *Quantization error on GIST1M base set.*

The proposed methods are notably inferior to the existing approaches on GIST1M data. Residual pyramid encoding achieves only slightly lower quantization error than PQ, regardless of the initialization. Residual pyramid encoding attains better results, but can barely compare with OPQ and AQ regardless of the code length.

Small-scale ANN experiments on GIST1M subsets confirmed that the recall is correspondingly lower for proposed methods. Search performance of pyramid encoding with random initialization AQ-P, as well as residual pyramid encoding with residual initialization AQ-RP(rs) are reported. These were found to be representative, as the initialization had a negligible effect on the search. Table 5 contains the relevant results.

Table 5. *Recall@T values on GIST1M*

64 bits (M = 8)	**R @1**	**R @10**	**R @100**	**R @1000**	**32 bits (M = 4)**	**R @1**	**R @10**	**R @100**	**R @1000**
PQ	0,076	0,218	0,504	0,8582	**PQ**	0,023	0,0675	0,1756	0,5045
OPQ	0,118	0,334	0,715	0,9465	**OPQ**	0,054	0,1419	0,3964	0,7905
AQ-B	–	–	–	–	**AQ-B**	**0,069**	**0,189**	***0,4666***	***0,809***
OCKM	***0,13***	***0,3579***	***0,7197***	–	OCKM	–	***0,172***	**0,4683**	**0,816**
CQ	**0,1352**	**0,3765**	**0,7294**	**0,972**	CQ	–	–	–	–
LOPQ	0,116	0,33	0,656	0,918	LOPQ	0,049	0,13	0,362	0,71
SQ	0,1148	0,3153	0,6983	***0,9559***	SQ	***0,0585***	0,1599	0,43	0,795
OTQ	–	–	–	–	OTQ	–	–	–	–
AQ-P	–	–	–	–	***AQ-P***	0,047	0,157	0,382	0,726
AQ-RP (rs)	0,097	0,268	0,563	0,88	***AQ-RP (rs)***	0,057	0,156	0,385	0,752

In the absence of reported AQ results for 64-bit codes, CQ maintains superiority with 8 codebooks, and would be likely to outperform OCKM on 32-bit codes, if the corresponding information was available. Both suggested approaches demonstrate relatively poor performance. They also do not benefit as much as others from the increase in number of codebooks, failing to surpass even OPQ with 64-bit codes.

5.5 Analysis of the results

Pyramid encoding on SIFT1M dataset demonstrates impressive performance. The earlier observation about the importance of codebook diversity is now supported with experimental evidence. Orthogonal initialization of AQ-P allows it to achieve significantly lower quantization error, especially with shorter codelengths. This property may prove beneficial, if the quantizer is used purely for data compression or another similar purpose.

The ANN performance of AQ with pyramid encoding is either comparable with or higher than other recent methods. Even with longer codes, when other AQ derivatives surpass the AQ-P in terms of recall, it stays competitive due to its mild computational requirements. The only quadratic components of AQ-P complexity are number of codebooks M, which is limited in scale and is rarely above $M = 16$, and search depth H, which can be set to a number as low as $H = 64$.

Ultimately the good performance of pyramid encoding on SIFT1M can be explained with an internal structure of the data. SIFT descriptors are obtained by concatenating 16 histograms of 8 bins each (thus 128 dimensions) [44]. Resulting vectors are thus very suitable for a straightforward decomposition into 4, 8 and 16 subspaces, which was one

of the major factors in product quantization success [18]. Referring to the theoretical foundations of OPQ (see Chapter 3.2.3), one can observe that for SIFT descriptors the variances of subspaces would be already roughly balanced, as well as independent (since elements may come from different histograms). When applied to SIFT data, OPQ finds a more optimal decomposition via rotation, reducing the existing inter-subspace dependence and balancing the variance further. AQ, in turn, drops the subspaces assumption entirely, solving a much harder problem in virtually unconstrained space. The pyramid encoding with orthogonal initialization starts from the PQ solution and then gradually relaxes the subspace constraints. The codebooks are slightly "perturbed" during the iterative learning procedure and cease to be strictly orthogonal, which allows for a significantly lower quantization error.

GIST descriptors have a completely different structure; each sample of GIST1M dataset is composed of three 320-dimensional subvectors, calculated from separate planes of RGB colorspace [18]. Each subvector is in turn comprised of a number of normalized orientation histograms with varying number of bins. This makes the "natural" PQ subspace decomposition of GIST significantly less efficient, especially considering that the histograms, unlike in SIFT, are global and share no spatial relation. Every histogram also represents a separate image property, meaning that subspace variances are less likely to be balanced. For this reason the performance of PQ on GIST1M dataset is quite poor.

Same conclusion, it seems, can be extended to the orthogonal initializations of proposed encoding schemes. Since both pyramid encoding and residual pyramid encoding are inherently local in their optimization approach, they fail to improve from the poorly chosen starting point. Random initialization, in turn, suffers from lack of codebook diversity, regardless of the data origin. Residual and hybrid initializations slightly alleviate the situation, but not enough to compete with more optimal subspace decomposition of OPQ. AQ avoids the issue, as its Beam Search optimization process is much less greedy; it is therefore more global and is able to find a better solution even from a random initial guess.

6. CONCLUSIONS AND FUTURE WORK

The original intent behind the pyramid encoding was to reduce computational costs and make AQ more practical, achieving a compromise between complexity and search performance. This goal was achieved, although not to a perfect degree. As the experiments demonstrated, pyramid encoding is indeed capable of performing on par or better than the other methods, but is ultimately dependent on the suitability of the initial guess for a given data distribution. Meanwhile, the complexity of pyramid encoding is among the lowest of the methods with comparable quantization error and recall. It is also clear that there is room for further analysis and improvement, but careful consideration must first be given for the current results.

The limitations of pyramid encoding stem directly from the assumptions it is based on. Random initialization, unlike in Beam Search, suffers from the local nature of the optimization. Codebook uniformity implies redundancy, meaning that the hierarchical merging cannot provide a sufficient error decrease. Since the codebooks can only be adapted to a specific database representation (codes) during training, there is no opportunity for them to improve. Orthogonal initialization is the opposite extreme: as the codebooks represent non-overlapping subspaces of the data, no redundancy is possible. Unfortunately, the encoding is not capable of mitigating the suboptimality of space decomposition, resulting in subpar performance.

Residual Pyramid encoding, despite its plausible formulation, fails to improve upon the pyramid encoding, especially considering its higher computation costs. If the residual initialization is used, the algorithm is outperformed even by randomly initialized pyramid encoding. Since the residuals are calculated from the uniformly initialized codebooks, they are naturally uniform themselves. As no further constraint is enforced during the training, the learned codebooks are not distinguishable from the randomly initialized ones. Hybrid encoding can potentially produce diverse codebooks, but capturing the residual of PQ seems to be inferior to simply applying PQ with larger M.

Residual Pyramid encoding has yet another drawback when compared to pyramid encoding. For larger numbers of codebooks ($M > 4$) residual pyramid encoding suffers from fixed PQ-like subspace decomposition, severely limiting the benefits of better representation. The computational benefits of linear scaling with respect to M are thus mostly irrelevant, as search performance improvement is minor.

6.1 Future work

To further improve upon the pyramid encoding, two general approaches may be considered. A conservative approach would be to make changes in the encoding, such as providing new initializations, introducing new constraints and or altering the pyramid structure. Alternatively, one can change the codebook adaptation step as well; this would mean altering the overall training scheme, leading to an effectively new quantizer that cannot share the same name with AQ.

Optimal rotation can be considered a logical step to improve upon the current results. It has contributed to the success of several predecessor methods (e.g. OPQ) and has a sound theoretical basis. Since OPQ initialization was not regarded as beneficial, it is reasonable to assume that the rotation matrix has to be adapted iteratively during training. While optimal rotation can never lead to increase of quantization error, it also increases complexity, especially in extremely high-dimensional cases. Training convergence can also be expected to proceed slower, due to currently learned codebooks being partially invalidated after the data transformation. Experimental assessment is necessary to decide if this line of work is worth pursuing.

If the adaptation procedure is changed, some of the previously discussed pyramid encoding drawbacks can potentially be avoided. For instance, the codebook recalculation can be integrated into the pyramid ascent, performed on each level separately. The quantizer learned with such an approach would significantly differ from AQ, as codebooks are no longer equivalent to each other during the adaptation step. This "fine-grained" control over codevector learning has an immediate benefit – the diversity of codebooks would be easier to maintain. Moreover, in this case the total number of operations performed throughout the complete training phase is not expected to change much. The computational costs of such an approach would thus remain well below the upper bound, imposed by AQ.

Another possible encoding option, which was briefly touched upon in Chapter 4.1, is the angle-based quantization. The whole database of vectors can be decomposed to norms (scalar values) and directions (unit vectors). The former can be efficiently quantized as a set of scalars, while the latter can be processed with the pyramid encoding. As outlined in Chapter 4.1, this transformation can simplify the formulations, reducing the full vectors to the cosines of angles between them. This problem can be easier to solve than the original one. Consider, for example, that the codebook diversity requirement in such case would amount to simply keeping codevectors with different orientations.

As was mentioned previously, a number of issues common to many quantization-based ANN methods stem from the local nature of the optimization. While locality and greediness are necessary to cope with otherwise highly complex encoding problems, there is a family of optimization techniques aimed at such tasks – a *metaheuristics family* [48].

It encompasses random search, simulated annealing, swarm optimization, genetic algorithms and a number of other approaches that solve the black-box problems, relying only on the objective function evaluations. While most metaheuristics are known for slow convergence and high computational costs, they are also capable of finding a global solution on extremely difficult fitness landscapes, where the local search fails. Additional research is required to determine the suitability of these methods to the encoding, as well as to find the ways to apply the domain-specific knowledge. It is possible that the global optimization would be capable of locating much better solutions for both the codebooks and the database codes, justifying the increase in computation.

Finally, there are more application areas to explore with described methods. For example, additive quantization can be used for approximate calculation of dot products between a given vector (equivalent to query) and a large number of other vectors (equivalent to database). This approach may prove useful when numerous linear models need to be estimated, or in large-scale kernel methods. Alternatively, since codebooks and assignments are learned in an unsupervised manner, analysis of the resulting quantizer can potentially provide insight into the data’s internal structure; tasks like clustering or dimensionality reduction are possible, given specific formulations.

REFERENCES

[1] C. C. E. Leiserson, R. R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to Algorithms, Third Edition*. 2009.

[2] J. Clausen, "Branch and bound algorithms-principles and examples," *Dep. Comput. Sci. Univ. Copenhagen*, pp. 1–30, 1999.

[3] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mob. Networks Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[4] D. Bollier, *The Promise and Peril of Big Data*. 2010.

[5] R. Weber, H. J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *Proc. 24th VLDB Conf.*, vol. New York C, pp. 194–205, 1998.

[6] C. D. Manning, P. Raghavan, H. Schütze, and others, *Introduction to information retrieval*, vol. 1, no. 1. Cambridge university press Cambridge, 2008.

[7] Y. Jing and S. Baluja, "Visualrank: Applying pagerank to large-scale image search," *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 30, no. 11, pp. 1877–1890, 2008.

[8] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Y. Zomaya, S. Foufou, and A. Bouras, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *Emerg. Top. Comput. IEEE Trans.*, vol. 2, no. 3, pp. 267–279, 2014.

[9] G. Shakhnarovich, P. Indyk, and T. Darrell, *Nearest-neighbor methods in learning and vision: theory and practice*. 2006.

[10] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*, Springer, 2007, pp. 325–341.

[11] L. Devroye, L. Györfi, and G. Lugosi, *A probabilistic theory of pattern recognition*, vol. 31. Springer Science & Business Media, 2013.

[12] S. Garcia, J. Derrac, J. R. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 34, no. 3, pp. 417–435, 2012.

[13] A. Iosifidis and M. Gabbouj, "Nyström-based approximate kernel subspace learning," *Pattern Recognit.*, vol. 57, pp. 190–197, 2016.

[14] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for Similarity Search: A Survey," *arXiv:1408.2927*, pp. 1–29, 2014.

[15] E. C. Ozan, S. Kiranyaz, and M. Gabbouj, "M-PCA Binary Embedding for Approximate Nearest Neighbor Search," in *2015 IEEE Trustcom/BigDataSE/ISPA*, 2015, vol. 2, pp. 1–5.

[16] R. M. Gray, “Vector quantization,” *ASSP Mag. IEEE*, vol. 1, no. 2, pp. 4–29, 1984.

[17] R. M. Gray and D. L. Neuhoff, “Quantization,” *Inf. Theory, IEEE Trans.*, vol. 44, no. 6, pp. 2325–2383, 1998.

[18] M. Douze, C. Schmid, H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search.,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, Jan. 2011.

[19] T. Ge, K. He, Q. Ke, and J. Sun, “Optimized product quantization,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 744–755, 2014.

[20] E. C. Ozan, S. Kiranyaz, and M. Gabbouj, “K-Subspaces Quantization for Approximate Nearest Neighbor Search,” *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1722–1733, Jul. 2016.

[21] A. Babenko and V. Lempitsky, “Additive Quantization for Extreme Vector Compression,” *Cvpr*, pp. 931–938, 2014.

[22] T. Zhang, J. Wang, and J. M. Com, “Composite Quantization for Approximate Nearest Neighbor Search,” *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32, pp. 838–846, 2014.

[23] A. Babenko and V. Lempitsky, “Tree Quantization for Large-Scale Similarity Search and Classification,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[24] C. Martinez, “Partial quicksort,” in *Proc. 6th ACMSIAM Workshop on Algorithm Engineering and Experiments and 1st ACM-SIAM Workshop on Analytic Algorithmics and Combinatorics*, 2004, pp. 224–228.

[25] P. Indyk and R. Motwani, “Approximate nearest neighbors: towards removing the curse of dimensionality,” in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, 1998, pp. 604–613.

[26] J. H. Friedman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[27] S. M. Omohundro, *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[28] P. N. Yianilos, “Data structures and algorithms for nearest neighbor search in general metric spaces,” in *SODA*, 1993, vol. 93, no. 194, pp. 311–321.

[29] S. Meiser, “Point location in arrangements of hyperplanes,” *Inf. Comput.*, vol. 106, no. 2, pp. 286–303, 1993.

[30] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma, “A survey of content-based image retrieval with high-level semantics,” *Pattern Recognit.*, vol. 40, no. 1, pp. 262–282, 2007.

[31] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 253–262.

[32] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in *Advances in neural information processing systems*, 2009, pp. 1753–1760.

[33] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, “Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, 2013.

[34] S. Graf and H. Luschgy, *Foundations of Quantization for Probability Distributions*, vol. 1730. Springer Berlin Heidelberg, 2000.

[35] A. Oliva and A. Torralba, “Modeling the shape of the scene: A holistic representation of the spatial envelope,” *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001.

[36] H. Jegou, H. Harzallah, and C. Schmid, “A contextual dissimilarity measure for accurate and efficient image search,” in *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on*, 2007, pp. 1–8.

[37] M. Norouzi and D. J. Fleet, “Cartesian K-Means,” *2013 IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 3017–3024, 2013.

[38] H. Jégou, M. Douze, C. Schmid, and P. Pérez, “Aggregating local descriptors into a compact image representation,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 3304–3311.

[39] Y. Kalantidis and Y. Avrithis, “Locally optimized product quantization for approximate nearest neighbor search,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2321–2328.

[40] G. F. Cooper, “The computational complexity of probabilistic inference using Bayesian belief networks,” *Artif. Intell.*, vol. 42, no. 2, pp. 393–405, 1990.

[41] J. Besag, “On the statistical analysis of dirty pictures,” *J. R. Stat. Soc. Ser. B*, pp. 259–302, 1986.

[42] L. Amsaleg and H. Jégou, “Datasets for approximate nearest neighbor search,” 2010. [Online]. Available: http://corpus-texmex.irisa.fr/. [Accessed: 14-Mar-2016].

[43] L. der Maaten and G. Hinton, “Visualizing High-Dimensional Data Using t-SNE,” *J. Mach. Learn. Res.*, vol. 9, no. 2579–2605, p. 85, 2008.

[44] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, 1999, vol. 2, pp. 1150–1157.

[45] J. Martinez, “Stacked Quantizers,” 2014. [Online]. Available: https://github.com/jltmtz/stacked-quantizers. [Accessed: 08-Apr-2016].

[46] J. Wang, J. Wang, J. Song, X. Xu, H. T. Shen, and S. Li, “Optimized Cartesian K-Means,” *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 180–192, 2015.

[47] J. Martinez, H. H. Hoos, and J. J. Little, “Stacked Quantizers for Compositional Vector Compression,” *arXiv Prepr. arXiv1411.2173*, 2014.

[48] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013.